1996

A tool for design and analysis of protocols

Viswanathan Narain
University of Wollongong

A TOOL FOR DESIGN AND ANALYSIS OF PROTOCOLS

A thesis submitted in partial fulfilment of the

requirements for the award of the degree

Master of Science (Honours)

from

UNIVERSITY OF WOLLONGONG

by

Viswanathan Narain, B.E

Department of Computer Science

1996

Acknowledgments

I would like to express my gratitude to my supervisors Dr. Rei Safavi-Naini and Dr. Neil Gray for guiding me on this project. I am indebted to them for the timely help and constant support rendered by them at every stage of the project.

I would also like to thank all members of the Centre for Computer Security Research for the invaluable help given by all of them. I am particularly grateful to Mansour Esmaili for helping me in making the thesis more presentable.

Last but not the least, I am thankful to my parents, without whose blessings and constant support, I would never have got this opportunity to engage in further studies in Computer Science.

Abstract

The existence of strong cryptoalgorithms is not sufficient to guarantee the security and/or authentication required in a system. To obtain an assurance for the security and/or authentication required, the underlying cryptoalgorithm must be used within a set of rules or procedures known as a protocol. Even if the cryptoalgorithm were secure, it is possible to subvert the protocol. There are numerous instances of failures in protocols employing public key and private key cryptoalgorithms. Several systems exist for analysis of protocols, to find such failures in them. These systems use formal analysis methods based on a model of communicating state machines or modal logics of belief like BAN and GNY. Besides, it is difficult to understand the working of protocols by looking at the description of messages in them.

This thesis proposes a graphical user interface based tool for aiding in the design and analysis of cryptographic protocols. It provides a feature to display the working of protocols in the form of a schematic diagram, as an educational tool for understanding protocols. Secondly, it provides an interface to an existing program for performing automated GNY logic analysis. The tool, we propose also provides facilities for single stepping through protocols whilst viewing the beliefs attained by the principals, viewing of proofs derived from the GNY tool, and a feature for run-time modification of initial assumptions. It has been used to perform analysis of well known protocols and the expected results are obtained for them.

Contents

List of Figures

Chapter 1

Introduction

1.1 Aim

The aim of this thesis is to describe the development of a graphical user interface (GUI) based tool to assist in the design and analysis of cryptographic protocols. The use of this tool is basically two-fold; to serve as an educational tool for understanding of cryptographic protocols and to provide a user friendly graphical user interface to perform analysis of protocols using the GNY logic, proposed by Gong, Needham and Yahalom.

1.2 Scope

The tool can be used to demonstrate the working of cryptographic protocols and helps with their security analysis. Cryptographic protocols [4, 7] are usually illustrated by showing principals as nodes and messages transmitted between them as arrows from the senders to the receivers accompanied by the text of the actual transmitted message. The tool presents protocols diagrammatically in this manner.

In GNY logic analysis, the *idealized* form of a protocol together with the *initial assumptions* of the principals are used to calculate a set of beliefs for each principal. By observing the set of beliefs, it can be inferred whether the protocol has achieved its goal or not.

The *Protocol Analyzer* provides aid in GNY logic analysis of protocols. It displays a given protocol in graphical form: the principals are shown as circular nodes and arrows between them represent potential communication channels between them. For each message in the protocol the link between the sender and the receiver is highlighted with an arrow, showing the direction, and the message content is displayed.

Given the concrete description of a protocol and a set of initial assumptions, the *Protocol Analyzer* performs GNY logic analysis as it steps through the protocol.

It displays the beliefs set of each principal separately at every stage in the protocol and provides a facility to modify the initial assumptions of the protocol and rerun the protocol. Thus one can observe the evolution of beliefs for each principal and also infer about the initial assumptions which are critical to the protocol achieving its goal at the end.

To implement the above features, the tool relies on previous work which is described in the next section.

1.3 Previous Work

The *Protocol Analyzer* proposed in this thesis relies on an existing tool for GNY logic analysis of protocols [12, 13, 29].

A basic tool for performing belief based BAN logic analysis of protocols is described in [12]. This is a tool written in XSB Prolog which accepts the input

specification of a protocol in BAN logic, along with the initial assumptions of all principals, and derives the beliefs attained by all principals due to the run of the protocol.

The tool is subsequently enhanced to perform GNY logic analysis of protocols [13, 31]. The enhanced version accepts the protocol specification and initial assumptions and derives the beliefs held by the principals, using GNY logic postulates.

The GNY logic tool just described is not a user friendly tool and expects the protocol messages to be in idealized form. Hence a manual transformation from the concrete form to the idealized form has to be performed . Besides, there is no way to look at beliefs of each principal at the end of each step, as all the beliefs obtained as a result of a step, or steps, are output by the tool, collectively. Also the beliefs are output in idealized form without any modifications, making them difficult to understand.

1.4 *The Protocol Analyzer*

The *Protocol Analyzer* described in this thesis, uses the GNY tool mentioned above, and provides a user friendly interface to it so as to overcome some of the limitations mentioned above.

The *Protocol Analyzer* takes the concrete description of a protocol and provides a translator for this language to the idealized form required by the GNY tool. It communicates with the GNY tool, obtains beliefs held by principals and displays them. Additionally it displays the user chosen protocol in graphical form.

1.5 Outline of the thesis

The thesis is organized as follows:

1. Chapter 2 describes security analysis of cryptographic protocols and illustrates typical failures in them.

2. Chapter 3 describes formal analysis of cryptographic protocols and specifically two belief based logics BAN and GNY.

3. Chapter 4 describes automation of protocol analysis and gives comparison of an existing GUI based protocol analysis tool, *The Interrogator*, and the automated GNY logic tool, described in the thesis.

4. Chapter 5 describes an overview of the features and design of the proposed *Protocol Analyzer*.

5. Chapter 6 includes details of results obtained by using the tool for analyses of Voting and Otway-Rees protocols.

6. Chapter 7 concludes the thesis by proposing possible extensions to the *Protocol Analyzer*.

Appendix A lists the source files for the entire program and Appendix B gives the output screens for results obtained by analyzing Voting and Yahalom protocols.

Chapter 2

Security in Protocols

2.1 Introduction

The basic goals of cryptography are to provide secrecy and authentication in a system. To achieve these objectives, cryptographers have developed complex cryptoalgorithms. These algorithms are designed to be resistant to attacks by intruders with access to powerful computers.

However the mere existence of strong cryptoalgorithms is not sufficient to guarantee the required security and/or authentication in the system. The cryptoalgorithm must be used within a set of rules or procedures known as a *protocol*, to provide the security and authentication required. In a system which employs a protocol, if the underlying cryptoalgorithm were broken then the intended function of the protocol could be subverted. On the other hand it is *feasible* to subvert a protocol without impeaching or even eroding the security of the underlying cryptoalgorithm.

There are instances of subversion of key distribution protocols, to distribute keys to unintended recipients, secrecy protocols, to publicly reveal contents of secret information, and digital signature protocols to make forgery possible

and yet all these are based on cryptoalgorithms that are considered secure. In the case of a protocol which uses Vernam encryption/decryption when used with a properly chosen one-time key is well known to be unconditionally secure [16] and still the protocol fails totally. This type of dramatic failure is known as a *protocol failure.*

In the case of many protocols, the security of the cryptographic portion is weakened by factors like reducing the size of the key space. Though such cases are potential sources of failure in the security functions of protocols, they are not the only sources of protocol failures. In the rest of this chapter, a range of protocol failures are considered and the reasons for such failures and conclusions that are derivable from them for protocol analysis and design are discussed.

2.2 Protocol Failures

2.2.1 TMN Protocol Failure

The first example considered is of a key distribution protocol whose goal is to provide a pair of principals, on an open communications channel, with a common cryptographic session key which is to be kept secret from eavesdroppers and other subscribers. This example is chosen because it illustrates how the protocol can fail by allowing an intruder to recover the session key in real-time, in spite of the protocol being based on one *unconditionally secure* [16] and one *provably secure* [16] cryptoalgorithm.

The protocol is that proposed by Tatebayashi, Matsuzaki and Newman [17]. The setting of the protocol is as follows.

It is assumed that all user terminals have the capability to carry out one of a complementary pair of public-key operations. An example is forming a

modular cube with respect to a composite modulus and the complementary operation being, taking a modular cube root. Another capability the terminals have is to carry out a simple symmetric key encryption (for example implementing unconditionally secure Vernam ciphers by forming the exclusive-OR of a binary one time key with binary text).

The protocol works as follows. A subscriber A wishing to communicate securely with subscriber B sends a randomly chosen one-time key encrypted under its end of the server's public key system with a request to set up a secure channel with B. The server requests B to generate a *random* session key. B does this and also encrypts it under the other end of the server's public key system and sends this cipher to the server. The server can perform the hard task of decrypting ciphers in the public-key system and decrypts these two ciphers to recover the session key generated by B and the one-time key generated by A. Ten, the server performs Vernam encryption of the session key and the one-time key by forming the exclusive-OR of their binary representations and sends the resulting cipher to A. A can then decrypt this cipher using the initially chosen one-time key, to obtain the session key. A and B now have a common key which they can use to communicate securely. The goal of the protocol seems to have been achieved. Terminals with very limited computational capability can exchange keys using two secure cryptoalgorithms - Vernam encryption with an appropriately generated one-time key which is unconditionally secure and extracting modular cube roots with properly chosen prime factors in the modulus which is provably secure [16]. In spite of this, the TMN protocol exhibits a simple failure.

The Failure

Let it be assumed that another subscriber C has eavesdropped on the three ciphers - the session key and the one-time key each encrypted using the server's

public key and the Vernam encryption of the session key with the one-time key in the setting up of a secure channel between A and B. C has an arrangement with another subscriber D such that D will return a session key when requested by the server, which is known to C. Even if D chooses the key randomly, C will know what key to expect.

When C sees the cipher $r_B^3 \ mod \ n$ pass from B to the server, C chooses a random number r_C and forms a cipher

$$r_C^3(r_B^3 \ mod \ n) \ mod \ n$$

and sends it to the server with a request to the server for a secure channel with D. In the meantime C precomputes r_C^{-1}, for later use. The server then requests D to supply a session key, which is r, is known to both C and D and this is undetectable to the server. The server takes the cube root of the above cipher to obtain $r_C r_B$ and forms the cipher

$$r \oplus r_C r_B$$

believing it to be the encryption of the session key supplied by D with C's one-time key and sends it to C. Knowing r, C can calculate $r_C r_B$ as:

$$r_C r_B = r \oplus (r \oplus r_C r_B)$$

Using the previously computed r_C^{-1}, r_B can be recovered as:

$$r_B = r_C^{-1}(r_C r_B) = (r_C^{-1}(r \oplus (r \oplus r_C r_B)))$$

Using r_B, C can now eavesdrop on all communication between A and B; without being detected. All this has happened in the time required to set up a channel on the net, undetected by the server.

The reason for this failure in the TMN protocol is basically because the session keys and one-time keys sent by the users to the server are assumed to be

random. The keys are to be chosen from some known or specified set or range according to a known probability distribution. Depending upon the type of protocol, this assumption may be verifiable and/or enforceable. Implicitly, it is known that the probability that the random number chosen by a user will be known to someone else is the probability that if each were to choose randomly from the same set using the same probability distribution, they would get the same value. Thus the assumption about what *random* means is neither verifiable nor enforceable. Besides, there is no method to prevent anyone from divulging anything they know with anyone they trust.

As a result one inference that can be derived for protocols is: In any protocol that calls for the generation of a random number, it is essential in the analysis of the protocol to determine whether there are deceptions that could be either carried out or furthered if the random value were shared with one or more of the other participants - but in secret from some of them.

2.2.2 Low Exponent Protocol Failure

The example considered here is that of a protocol using RSA [18]. It uses a small public key exponent in order to accomplish fast and cost-effective encryption operations.

The general environment of this protocol is a large communications network in which the messages transmitted between two users should not be readable to other users. In the protocol, the i^{th} user has to choose two large primes p_i and q_i and publish their product n_i as the modulus for the RSA algorithm used for communication with him. An encryption/decryption pair $\{ e_i, d_i \}$ is chosen and one of these, e.g., d_i is published. If the encryption exponent chosen is a small integer, as can very well be the case as then the implementation is quicker and simpler, this can cause the protocol to fail as shown below.

If the exponent is d and the same message M is sent to d users, then the protocol is susceptible to failure. This is illustrated in the case when d = 3. Suppose that user 1 whose public exponent is 3 decides to send a message M to users 2, 3 and 4. The ciphertexts in those cases are:

$$\begin{aligned} C_2 &= M^3 \, mod \, n_2 \\ C_3 &= M^3 \, mod \, n_3 \\ C_4 &= M^3 \, mod \, n_4 \end{aligned}$$

In the case when n_2, n_3 and n_4 are relatively prime, the Chinese Remainder theorem will enable the calculation of M^3 mod (n_2, n_3, n_4) from C_2, C_3, C_4. But since $M^3 < n_2 n_3 n_4$, M's value can be recovered. If n_2, n_3, n_4 are not relatively prime, then attacks from the common modulus protocol apply. The various types of attacks on an RSA based protocol employing a common modulus are given in [8]. Thus it can be seen that even an intruder has enough information to recover the message.

A method to salvage the protocol from this attack is to never send exactly the same message. For this an additional variable like a timestamp is concatenated to the message before encrypting it. Using the scheme, the modified ciphertexts from the above example would be:

$$\begin{aligned} C_2 &= (2^{|t_2|}M + t_2)^3 \, mod \, n_2 \\ C_3 &= (2^{|t_3|}M + t_3)^3 \, mod \, n_3 \\ C_4 &= (2^{|t_4|}M + t_4)^3 \, mod \, n_4 \end{aligned}$$

where t_2, t_3 and t_4 are the timestamps associated with each message. However Hastad [19] has shown that even this may not vary the plaintext enough

to overcome the weakness in this protocol. Hastad showed in his paper that a system of modular equations

$$p_i(x) = 0 \ mod \ n_i, \ 1 \leq i \leq k$$

of degrees no greater than d can be solved in polynomial time if the number of equations is at least d(d+1)/2. In the case with an exponent of 3 if the message is sent to a minimum of 7 users of the network, the message may no longer be a secret to an intruder. However the timestamps must be known and this is assumed as they can be estimated before Hastad's algorithm is applied as they form a small number of bits in the entire message bit string.

The above protocol failure emphasizes the need for the designer of a protocol to consider information which can be gained from a collection of ciphertexts whose plaintexts are related (in this case equal or differing only by timestamps) or whose keys are related (in this case the same exponent with relatively prime moduli).

2.2.3 The Low Entropy Protocol Failure

The two protocol failures considered so far were mainly due to some mathematical properties of the cryptoalgorithms used in them. However this is not the reason for all possible failures. We show an example of a failure which is not dependent upon the cryptoalgorithm used.

In a public key system being used to provide a secrecy channel for a protocol, the encryption key is publicly known so any message encrypted with it can only be understood by the intended recipient who possesses the private decryption key. But if the number of valid and meaningful messages is small, then an opponent could precompute the encryption of those messages. Then when an encrypted message is sent through the channel, it is sufficient

to search through the table of precomputed values to establish the meaning of the message. The complexity of the above task depends directly on the size of the message space. The message space must be large enough to preclude an attack in which the set of all messages is pre-encrypted by an opponent who can intercept a ciphertext and recover the corresponding plaintext by exhaustive search.

However the encryption of all messages in the space is not required. If the ciphertext for a significant portion of the space is precomputed, the meaning of a given ciphertext can be discerned by simple pattern matching of the ciphertext with the precomputed table values, whenever possible, without decrypting the rest of the message. To establish what the term *significant* in this context means, we use the term *entropy. Entropy* is a measure of uncertainty in the message space. If the message space has low entropy, small amount of information about a given message is enough to reveal the message.

In their paper Holdridge and Simmons [20] show a protocol failure in an application proposed by Bell Telephone Laboratories for use in secure telephony [21, 22]. This system is based on public key encryption for subscribers on a mobile radio telephone network. Public encryption keys of all subscribers are stored in a public key directory whereas their decryption keys are kept secret. The communication from one subscriber to another is done as follows. A sampled and digitized speech signal of the sender is encrypted with his encryption key from the public directory. The receiver uses his private decryption key to obtain the plaintext message.

The problem with this system is that voice signals have a rather narrow bandwidth, thus enabling attacks on them. Redundancy of the language, and the the inter-symbol and inter-word dependencies are useful clues in ascertaining the meaning of a sample of corrupted speech signals. Thus the protocol

fails to provide the required secrecy.

In the case of this failure, the public key algorithm used has no role in the failure. The underlying cryptosystem was not broken, but still the secrecy channel failed. It is merely a failure of the protocol, which is the use of a public key encryption system to protect messages drawn from a message space with *low entropy*, to provide the required privacy.

2.2.4 Single Key Protocol Failure

All the protocol failure cases considered so far have used public key algorithms. To show that even conventional single key encryption systems are prone to protocol failures, we consider a protocol using DES [23], which fails to provide the required authentication.

Message Authentication Codes and *Manipulation Detection Codes* have been used to convince the receiver of a message that indeed the message was not manipulated by an intruder during its transmission over an insecure channel. A message authentication code (MAC) uses a modified encryption function with a secret key different from that used to encrypt the message. It is typically small so as to keep the size of the total message to the minimum. A manipulation detection code (MDC) uses a function with no secret key.

A protocol for protecting information using DES with an MDC was proposed for inclusion in a federal data communications standard [24]. It was simple and appealing until it was discovered that it failed to detect several manipulations.

The protocol works as follows. The data to be transmitted is divided into n blocks of k bits each, where k varies between 1 and 64. Let these plaintext blocks be X_1, X_2, ... X_n. Ciphertext blocks Y_1, Y_2, ... Y_n are formed from them using a suitable mode of DES, e.g., one of the chaining modes of DES. An

MDC Y_{n+1} is formed which is the exclusive-OR sum of the n plaintext blocks. This block is used by the receiver to verify that the data received were indeed not tampered with during transmission. The receiver decrypts the data and computes an exclusive-OR sum of them and compares the result to the MDC. If they agree, then he will conclude that the ciphertext was not manipulated.

The failure in this protocol is illustrated using the Cipher Block Chaining mode of DES [8]. In this mode a 64 bit initialization vector Y_0 and a 56 bit key are exchanged secretly between the communicants. The n ciphertext blocks Y_1, Y_2, ..., Y_n each of 64 bits are calculated as:

$$Y_i = Y_{i-1} \oplus E(K, X_i)$$

where E(K, X_i) represents encryption with DES under key K of the message X_i. The MDC is calculated as:

$$Y_{n+1} = \oplus_{i=1}^{n} X_i$$

At the receiver's end the first n blocks Z_1, Z_2, ..., Z_n are decrypted using the reverse operation:

$$W_{i+1} = W_i \oplus D(K, Z_i)$$

where D(K, Z_i) represents decryption with DES under key K of the message Z_i. Then the exclusive-OR sum of these n blocks is formed and compared to Y_{n+1}. If they agree then the receiver would conclude that the message had not been manipulated. Since the check merely confirms that $\oplus\ X_i = \oplus\ W_i$, an intruder could modify the message say by interchanging some of the blocks, since their sum would still be the same.

Blocks can even be inserted into the message as long as they are done in pairs. This is because the exclusive-OR sum of a block with itself is zero

and therefore the MDC will not be affected. One example is of an intruder knowing an encrypted block of data that if added could change the value of a deposit from $1,000 to $1,000,000 in a message without the change being detected at the receiving end. Although the effect of such failures depends on the particular application, the basic inference is that the protocol has failed to provide adequate protection against undiscovered manipulation.

2.3 Summary and Further Analysis

A number of protocol failures have been surveyed in the previous section. The intention in studying them is to throw light on learning certain basic principles to be applied during the design of protocols. The distinction between the *breaking* of cryptosystems and *failure* of protocols is important. A protocol failure can lead to the definition of new guidelines for the use of a particular set or class of algorithms whereas a broken cryptosystem results in the rejection of the given algorithm from consideration by protocol designers.

With the help of protocol failures seen so far, some general principles for protocol analysis can be derived. They are described in detail as principles of cryptanalysis in [16]. They are outlined here as:

1. All the properties of all the quantities involved in the protocol must be carefully enumerated, including explicit assumptions in the protocol specification and implicit assumptions in the protocol setting.

2. No properties must be accepted unless accompanied by a proof to verify or enforce them. For each possible violation of a property, the protocol must be critically examined to see if this creates any difference in the outcome of the execution of the protocol. Combinations of parameters as well as single parameters must be considered.

3. In the final analysis, if the outcome of exercising the protocol can be influenced as a result of the violation of one or more assumed properties, the next step is to determine whether this fact can be exploited to cause a sensible deception. For example in the Diffie-Hellman key exchange protocol [25], it is possible to influence the outcome by violating the assumed properties of one or more of the parameters involved, but this does not result in any reasonable deception. A protocol failure is said to occur whenever the function of the protocol can be subverted as a result of any possible violations.

Similar conclusions are drawn for making security systems and protocols robust [27]. The essence of robustness in security systems is explicitness. Cryptographic protocols interact in ways that break security when their designers do not specify the required properties explicitly and protocol failures occur because naming, freshness and chaining properties are assumed implicitly to hold between two parties. However just making every security property about a system explicit is not a solution to building robust and hence failure resistant systems. The more aspects of any system are made explicit, the more information its designer has to deal with. This is equally applicable to design and evaluation of security systems.

The purpose of these guidelines is to clarify the setting which the protocol designer may have assumed would exist for the application of the protocol and also aid in the clarification of the purpose of each assumption or property, aid in identifying the most crucial elements in the protocol and make the modification of the protocol simpler to accomplish, if actually in implementation some assumption cannot be satisfied. These guidelines are not expected to prevent protocol failures by themselves, but help in the early detection of flaws.

In the following chapters we will see how these principles are incorporated

into the process of analysis and how they help in the overall process of protocol analysis and design.

Chapter 3

Formal Analysis of Protocols and GNY Logic

3.1 Formal Analysis Methods

As we have seen in the previous chapter, protocols which are not designed correctly are prone to attack and consequently they may not be able to provide the secrecy and/or authentication required of them. Such security flaws are subtle and hard to find and examples exist in literature to show that such flaws were not discovered for some time despite extensive manual analysis on the concerned protocols. One example is of the flaw in Needham-Schroeder conventional key protocol, in which an intruder masquerades as a genuine participant and passes an old compromised key as a good new one. Another example is of a protocol in the CCITT X.509 draft standard [10] for which Burrows, Abadi and Needham [4] showed that an intruder could make an old session key be accepted as a new one. There are numerous other undocumented examples.

These kinds of problems in protocols are well suited for the application

of *formal methods.* Due to the variety and counterintuitiveness of the flaws, an informal analysis may be too prone to error to be reliable. On the other hand, since the protocols are well contained, modeling and analyzing them is tractable enough so that formal methods can be applied. The use of formal methods in analysis of cryptographic protocols has become widespread since the beginning of the nineties, as more and more undiscovered security flaws were found using formal analysis techniques.

Formal methods in the analysis of cryptographic protocols have commonly followed two approaches :-

1. Methods based on communicating state machines

2. Systems based on modal logics of knowledge and belief

3.1.1 Methods based on communicating state machines

These methods are based on representing a protocol as a set of *communicating state machines*, one per each party in the protocol. The inputs and outputs of these machines are messages directed to each other. A party represented by a state machine can advance from one state to another due to an *event.* An event is caused by the sending or receiving of a message.

The starting point for most versions of the state machine approach has been the work of Dolev and Yao [6]. In the Dolev and Yao model, the network is assumed to be under the control of an intruder who can intercept all traffic, create, alter and destroy messages and perform any operations such as encryption that is available to legitimate users of the system. However, initially the intruder is assumed not to know any secret information such as encryption keys of honest users in the system. The goal of the intruder is to find a word that is meant to be secret.

The main drawbacks of the Dolev and Yao model are that it can be used only to detect secrecy failures and that it does not allow modeling of participants' behaviour from one state to another. Most of the tools that use this model use a mechanism to describe the behaviour of the protocol participants. One of the earliest systems to use the Dolev and Yao model is the *Interrogator* developed by Millen et al [7]. We will discuss this system in detail in the next chapter but stated briefly, the Interrogator attempts to find security flaws in protocols by an exhaustive search of the state space.

3.1.2 Systems based on modal logics

The other formal approach in the analysis of cryptographic protocols is to use modal logics similar to those that have been developed for the analysis of the evolution of knowledge and belief in distributed systems. In such a logic, there are various statements about belief in, or knowledge of, messages in a distributed system and inference rules which can be used to derive beliefs and/or knowledge from other beliefs and knowledge. One of the best known and widely used among such logics is the BAN logic, due to Burrows, Abadi and Needham [4].

In BAN, an initial set of beliefs is assumed. The beliefs are updated as a result of further messages sent and received in the protocol. Using the BAN inference rules, the set of beliefs is expanded to include beliefs that can be derived using the initial beliefs and the beliefs obtained as a result of the most recent message. Finally, if the set of beliefs includes belief statements representing the goal of the protocol, then the protocol is assumed to be secure. BAN logic is simple and easy to apply, but among the many drawbacks it has are that, it does not differentiate between seeing a message and understanding it. The logic assumes that all principals are honest and go by the rules of

the protocol and it does not model knowledge and therefore no results about secrecy can be derived, using it.

The approach taken by Gong, Needham and Yahalom [5] in the GNY logic is to increase the scope of BAN logic itself, in order to increase its effectiveness. This logic includes a notion of recognizability that allows a principal to reason about the content of messages it expects to receive, and constructs for establishing honesty of principals, among other improvements. A detailed discussion of the GNY logic can be found in section 2.3 of this chapter.

3.2 BAN Logic

BAN logic is a modal logic representing beliefs of participants in a protocol. Three types of objects are dealt with : *principals*, *keys* and *nonces*. Basically BAN logic builds upon statements about messages sent and received throughout the protocol. In an analysis of a protocol, an initial set of *beliefs* is assumed. *Beliefs* are statements about the current status of each principal or participant in the protocol. Each message received is mapped to another set of beliefs. Then the *Inference Rules* of BAN [4] are applied to derive a new set of beliefs attained from the initial beliefs and those gained by the message just conveyed. If the set of beliefs finally attained can convey that the protocol goal is achieved, then the protocol is assumed correct. If the set of beliefs is inadequate, then this suggests inadequacy of protocol statements or initial assumptions.

Since the conventional notation of protocol message representation is not convenient for manipulation in the logic, each message is transformed into a logical formula. This formula is annotated with *assertions*. An assertion usually describes beliefs held by the principals at the point in the protocol

where the assertion is inserted. A detailed description of basic constructs and inference rules can be found in [4]. Here we describe them briefly.

3.2.1 Formulae

P $\models$ X : P *believes* X. P believes that X is true.

P $\triangleleft$ X : P sees X. P has received a message containing X.

P $\mid\sim$ X : P once said X. P has sent a message containing X.

P $\Rightarrow$ X : P has jurisdiction over X. P is a trusted or delegated authority on X.

$\sharp$(X) : X is fresh. No message sent in the past contained X.

P $\overset{K}{\leftrightarrow}$ Q : K is a *shared key* between P and Q. No one except P or Q or someone they trust can know K.

$\overset{K}{\rightarrow}$ P : K is a public key of P. K^{-1}, the matching secret key will not be discovered by anyone other than P or someone trusted by P.

P $\overset{X}{\rightleftharpoons}$ Q : X is a secret known to P and Q or to principals trusted by them.

$\{X\}_K$: X is encrypted under key K. It is assumed that $\{X\}_K$ originates from a principal P.

$\langle X_Y \rangle$: X is combined with formula Y. This proves the identity of the originator of this formula.

3.2.2 Inference Rules

The Inference rules of the logic capture the basic principles used in the design of protocols. An inference rule is written as,

$$\frac{X_1, ..., X_n}{Y},$$

and serves to explain that if X_1, ..., X_n hold then Y is true.

There are three basic inference rules in the BAN logic :

1. **Message Meaning** - The message meaning rule for shared keys states that the identity of the sender of an encrypted message can be deduced from the encryption key used :

$$\frac{P \models Q \overset{K}{\leftrightarrow} P, P \triangleleft \{X\}_K}{P \models Q \mid\sim X} \tag{3.1}$$

with P $\neq$ R where P $\triangleleft \{X\}_K$ from R is implied.

2. **Nonce Verification** - This rule expresses the check that a message is recent and hence that the sender still believes in it :

$$\frac{P \models \sharp(X), P \models Q \mid\sim X}{P \models Q \models X} \tag{3.2}$$

that is, if P believes that X could have been uttered only recently and that Q once said X, then P believes that Q has said X recently, and hence that Q believes X.

3. **Jurisdiction** - This rule states that one must believe what a trusted principal believes in :

$$\frac{P \models Q \Rightarrow X, P \models Q \models X}{P \models X} \tag{3.3}$$

that is, if P believes that Q has jurisdiction over X and P believes that Q believes in X, then P believes in X.

3.2.3 Analysis

In the actual analysis, each protocol step is transformed into an idealized form. For instance, the concrete protocol step A $\rightarrow$ B : $\{A, K_{ab}\}_{K_{bs}}$ is idealized as A $\rightarrow$ B : $\{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$ and thus the formula B $\triangleleft \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$ holds, when the message is received by B. In the protocol analysis the following steps are followed:

- The idealized protocol is derived from the original one.
- Assumptions about the initial state are written.
- The goal of the protocol is written as a set of logical formulae.
- Logical formulae describing the state of the system are attached as assertions after each protocol statement.
- Logical postulates (Inference Rules) are applied to the assumptions and assertions to discover the beliefs held by the protocol participants.

The beliefs held by all the participants, obtained by applying the logical postulates to the assertions at the end of the last step help in determining whether the protocol has achieved its desired goal or not.

The protocol achieves its goal if the set of beliefs obtained after the last step of the protocol include the formulae specifying the goal of the protocol.

3.3 GNY logic

GNY logic can be seen as an extension of BAN. The GNY logic [5] is more expressive than BAN logic. It does not, for example assume that redundancy is always present in encrypted messages and instead it incorporates a new notion of *recognizability* which captures a recipient's expectation of the contents of messages to be received.

GNY logic distinguishes between what one possesses and what one believes in. This allows the content of the message and the information implied by it to be treated separately. Some of the existing notions in BAN are modified and made to correspond more naturally to execution states and are thus more intuitive. For example, plaintext formulae which are not considered to be

useful in BAN are treated similar to encrypted formulae in GNY as they can in some cases be used to derive further conclusions. The logic also has new notation to represent functions other than encryption, such as decryption and one way hash functions and new notation for denoting private keys associated with public keys. The GNY logic thus assists in the analysis of a wider range of protocols.

The three most notable notions which can be represented explicitly in GNY logic, are discussed below.

3.3.1 Possession

This notion allows reasoning at a finer level of detail than BAN logic. This notion is implicit in many BAN rules. Consider the message decryption rule for shared keys [4].

$$\frac{P \models P \overset{K}{\leftrightarrow} Q, P \triangleleft \{X\}_K}{P \triangleleft X}$$

It is implicitly assumed that P has K. There is a distinction in possessing a key and believing anything about it. In GNY, we have the construct P $\ni$ X, meaning P possesses X, to represent this fact.

3.3.2 Recognizability

In BAN, sufficient redundancy in messages is assumed so that decryption of a ciphertext with the correct key results in a recognizable message. This is not assumed in GNY logic and instead a construct P $\models \phi$(X) is introduced, which denotes that P believes that X is recognizable. This explicates the expectations of P about X before actual receipt of the message.

3.3.3 Honesty

BAN logic assumes that principals participating in a protocol are *honest.* It can be inferred from the nonce-verification rule wherein a principal who recently said a message is assumed to believe in the message. To make the notion of honesty explicit, GNY logic introduces a construct $P \models Q \Rightarrow Q \models *$ which means that P believes Q to be *honest* and *competent.*

Another feature GNY logic provides is that it dispenses with the separate notations for shared keys and shared secrets in BAN and provides a common syntax $P \overset{S}{\leftrightarrow} Q$ with the semantical interpretation that S is a suitable secret for P and Q.

Along with the three main notions, an important concept in GNY logic is beliefs about others' beliefs, which is briefly described below.

Beliefs about others' beliefs

Each principal in a protocol expresses his beliefs by sending messages and as such the beliefs held by the sender are preconditions for the message to be sent. A message of the same form may carry different meanings in different protocols, depending upon the context. If a receiver of a message believes that the sender is honest and competent then he may believe in the beliefs held by the sender. Therefore depending upon levels of trust, reasoning about beliefs can be achieved.

A complete list of GNY postulates can be found in [5].

3.3.4 Protocol Analysis Using GNY Logic

Protocols are typically described by listing messages sent between the principals and by symbolically showing the source, destination and contents of

each message. A simple transformation is required to attain a form suitable for manipulation in GNY logic. The steps required in the transformation are described below.

Not-originated-here formulae

The first step in the protocol analysis is related to the fact that a typical protocol description does not distinguish between X and $*X$. P $\triangleleft$ $*X$ means that P is not the first one to convey X in the current run of the protocol.This aspect is entirely implicit in the standard protocol description. In order to avoid a much more complex form of logic, a *parser* is used to insert the star's in a protocol description.

Parser

The parser has to perform the following operations. For each line in the description of the form A $\rightarrow$ B : X, if B = A, this results in an error condition; otherwise this statement is split into two statements : A $\mid\sim$ X and B $\triangleleft$ X. For each complete formula Y which forms part of the line B $\triangleleft$ X, if Y does not first appear in a line A $\mid\sim$ X in the protocol, a star in inserted before Y. The parser would also mark ($*$X, $*$Y) instead of a more compact $*$(X, Y). After this procedure, the parser drops all lines of the type A $\mid\sim$ X.

Message Extensions

The descriptions that specify when a particular principal should proceed, for example certain conditions being met or certain beliefs being held, are often given verbally along with the standard protocol description. A method is needed to represent such preconditions to a formula within the logic. *Message extensions* are defined for this purpose. In X $\rightsquigarrow$ C the precondition for the

formula X is represented by statement C which is a message extension. Hence the verbal explanations associated with protocol descriptions are translated into the logic and inserted after the formula. Message extensions are to be appended for every formula in a protocol message. The methodology of appending message extensions to messages will be discussed in the chapter 4, when its implementation is discussed. It is assumed for the sake of simplicity that a formula without a star prefix has no extension appended to it, as no new conclusions should be derivable from such a formula, which is already conveyed.

Annotated Assertions

A protocol is a sequence of *told* statements C_1, C_2, ..., C_n each of the form P $\triangleleft$ X. An annotation for a protocol consists of a sequence of assertions and conjunctions of statements, inserted before the first told statement and after each told statement. The first assertion contains the assumptions and the last contains the conclusions. The assertions are derived by syntactic application of GNY postulates [5] to statements. The assertions obtained at the end of a given protocol represent the final positions of all principals and these are used to determine if the protocol under consideration has achieved its goal or not.

3.4 Example of GNY Analysis

We now show an example of the GNY reasoning process by analyzing the Needham-Schroeder conventional key protocol. This protocol has influenced the design of a significant number of existing systems and published protocols. It also serves as one of the examples in the BAN logic paper [4]. We will see the differences between the BAN and GNY approaches and the advantages

that GNY offers.

The goal of Needham-Schroeder protocol is for two principals A and B to be provided with a shared secret key [1, 4], which would be subsequently used as a session key. A trusted *Authentication Server* S shares common secrets with all participants and can generate good quality session keys. The goal of the protocol is that at the end of the protocol, each principal should possess a session key and be convinced in the goodness of that key. Additionally, each principal may be required to believe something about the state of the other principal. This may include believing that the other principal possesses the key or that it believes in the validity of the possessed key.

The Needham-Schroeder protocol consists of the following messages :-

1. A $\rightarrow$ S : A, B, N_a

2. S $\rightarrow$ A : $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

3. A $\rightarrow$ B : $\{K_{ab}, A\}_{K_{bs}}$

4. B $\rightarrow$ A : $\{N_b\}_{K_{ab}}$

5. A $\rightarrow$ B : $\{N_b - 1\}_{K_{ab}}$

N_a and N_b are nonces for A and B respectively, K_{as} and K_{bs} are the secret keys between A and S and B and S respectively. K_{ab} is the session key for A and B generated by S. Message extensions are added to the above messages which serve to replace the verbal explanations accompanying the run of the protocol. F is used to denote the decrement computation. The following idealized description of Needham-Schroeder protocol is thus obtained using the GNY parser.

1. S $\triangleleft$: $*$A, $*$B, $*N_a$

2. A $\triangleleft$: $*\{N_a, B, *K_{ab}, *\{K_{ab}, A\}_{K_{bs}} \rightsquigarrow S \models A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{as}} \rightsquigarrow$ S $\models$ A $\overset{K_{ab}}{\leftrightarrow}$ B

3. B $\triangleleft$: $*\{*K_{ab}, *A\}_{K_{bs}} \rightsquigarrow$ S $\models$ A $\overset{K_{ab}}{\leftrightarrow}$ B

4. A $\triangleleft$: $*\{*N_b\}_{K_{ab}}$

5. B $\triangleleft$: $*\{*F(N_b)\}_{K_{ab}} \rightsquigarrow$ A $\models$ A $\overset{K_{ab}}{\leftrightarrow}$ B

The above idealized description is obtained by using the steps outlined in the previous section.

3.4.1 Protocol Analysis

We have the following initial assumptions :-

A $\ni K_{as}$, A $\ni N_a$

A $\models$ A $\overset{K_{as}}{\leftrightarrow}$ S, A $\models \sharp(N_a)$, A $\models \phi(N_a)$

B $\ni K_{bs}$, B $\ni N_b$

B $\models$ B $\overset{K_{bs}}{\leftrightarrow}$ S, B $\models \sharp(N_b)$, B $\models \phi(N_b)$

That is each principal possesses a secret key and believes it is a good key between himself and the authentication server. He also possesses a nonce and believes it to be fresh. In addition, A and B believe their nonces N_a and N_b to be recognizable, respectively.

A $\models$ S $\Rightarrow$ (A $\overset{K_{ab}}{\leftrightarrow}$ B), A $\models$ S $\Rightarrow$ S $\models *$,

A $\models$ B $\Rightarrow$ B $\models *$.

B $\models$ S $\Rightarrow$ (A $\overset{K_{ab}}{\leftrightarrow}$ B), B $\models$ S $\Rightarrow$ S $\models *$,

B $\models$ A $\Rightarrow$ A $\models *$.

A and B believe that S has jurisdiction over quality secrets to be shared between them. A and B also believe S to be honest and competent. Moreover, A believes that B is honest and competent and B believes that A is honest and competent. These two assumptions are not necessary but as will be seen

below, the added trust implied by these assumptions enables A and B to attain stronger final positions.

S $\ni K_{as}$, S $\ni K_{bs}$, S $\ni K_{ab}$

S $\models$ A $\overset{K_{as}}{\leftrightarrow}$ S, S $\models$ B $\overset{K_{bs}}{\leftrightarrow}$ S, S $\models$ A $\overset{K_{ab}}{\leftrightarrow}$ B

S possesses valid keys with A and B and he also believes that they are secret with A and B respectively. S also believes that K_{ab} is a suitable key between A and B.

Now, we apply the postulates to each message. The postulates of GNY used in this analysis are given for reference in the appendix to this chapter.

Message 1

Applying Being-told rule 1 (T1) and Possession rule 1 (P1), we obtain S $\ni$ (A, B, N_a), meaning S possesses A, B, and N_a.

Message 2

The extension to the message, S $\models$ A $\overset{K_{ab}}{\leftrightarrow}$ B is valid because it holds when the message is sent, as it is part of the initial assumptions. Also S is sure that the recipient A will be sure that K_{ab} is a key for himself and B (and no other principal) as B's name is also included in the message.

By applying Being-told rules 1 (T1) and 3 (T3) and Possession rule 1 (P1), we get

$$A \ni (N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}})$$

that is A possesses the contents of the message.

Applying Being-told rule 2 (T2), we get A $\ni K_{ab}$, that is A possesses the key K_{ab}.

Applying Freshness rule 1 (F1), we obtain

$$A \models \sharp(N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}})$$

A believes that the message is fresh and recently generated.

Applying Recognizability Rule 1 (R1), we get

$$A \models \phi(N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}})$$

that is A believes that the contents of the message are recognizable.

Applying Message Interpretation rule 1 (I1), we get

$$A \models S \mid\sim (N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}).$$

A believes that S once said the message.

Applying Jurisdiction rule 2 (J2), we obtain

$$A \models S \models A \overset{K_{ab}}{\leftrightarrow} B$$

A believes that S believes that K_{ab} is a good key between A and B.

Applying Jurisdiction rule 1 (J1), we obtain

$$A \models A \overset{K_{ab}}{\leftrightarrow} B$$

that is A believes that K_{ab} is a good key between A and B.

Message 3

It can be seen that the extension S $\models$ A $\overset{K_{ab}}{\leftrightarrow}$ B is valid.

Applying Being-told rules 1 (T1) and 3 (T3) and Possession rule 1 (P1), we obtain B $\ni K_{ab}$. B possesses K_{ab}.

We are not able to derive new beliefs from this message. Particularly, we cannot say that the contents of message 3 are fresh. For B, it would seem as if message 3 is a replay of a message from a previous run of the protocol. In BAN analysis of the above message, we could conclude that S once said the message, but we cannot conclude similarly in GNY analysis and so B is not sure that the message originated from S and cannot convince himself of S's beliefs and cannot make use of the extension to the message.

Message 4

Applying Being-told rules 1 (T1) and 3 (T3) and Possession rule 1 (P1), we obtain A $\ni N_b$. A possesses N_b.

No further conclusions can be derived. Although K_{ab} is used in the message for encryption, A does not know who sent the message or that B now possesses the key. The reason is N_b is a random number, not recognizable to A.

Message 5

None of the GNY postulates lead us to any beliefs or possessions from this message. B cannot even be convinced that the message originated from A, although possessing K_{ab}, the contents of the message are recognizable to B (Recognizability rule 1 (R1)). Finally, B is not convinced that he shares the key K_{ab} with A.

It can thus be concluded from our reasoning that given the first three messages of the protocol, the last two messages attain nothing of use and can thus be eliminated without weakening the final position attained, originally.

Summarising, we have the following conclusions :

A $\ni K_{ab}$, A $\models$ A $\overset{K_{ab}}{\leftrightarrow}$ B

B $\ni K_{ab}$

A possesses and believes in the secret key but B possesses the secret key but cannot believe in it. Neither A nor B believe anything about each other. As can be seen, the original goal of the protocol, that is A and B share a key K_{ab}, believe that it is suitable for them and mutually believe that the other principal also believes in its suitability, is not achieved.

3.5 Appendix

In this appendix, we list all the GNY logic postulates.

3.6 GNY logic rules

3.6.1 Rationality rule

This rule basically states that the set of GNY postulates can be expanded to permit reasoning about a principal's beliefs regarding the state of other principals.

If $\frac{\mathrm{C_1}}{\mathrm{C_2}}$ is a rule, then for any principal P, so is $\frac{\mathrm{P} \models \mathrm{C_1}}{\mathrm{P} \models \mathrm{C_2}}$.

3.6.2 Being-Told Rules

This set of rules help in determining whether a principal P is told a component of a message as a result of seeing a more complex message containing functions of that component.

T1 $\dfrac{P \triangleleft *X}{P \triangleleft X}$

T2 $\dfrac{P \triangleleft (X,Y)}{P \triangleleft X}$

T3 $\dfrac{P \triangleleft \{X\}_K,\ P \ni K}{P \triangleleft X}$

T4 $\dfrac{P \triangleleft \{X\}_{+K},\ P \ni -K}{P \triangleleft X}$

T5 $\dfrac{P \triangleleft F(X,Y),\ P \ni X}{P \triangleleft Y}$

T6 $\dfrac{P \triangleleft \{X\}_{-K},\ P \ni +K}{P \triangleleft X}$

3.6.3 Possession Rules

This set of rules aid in determining what a principal P possesses as a result of some other possession or possessions.

P1 $$\frac{P \triangleleft X}{P \ni X}$$

P2 $$\frac{P \ni X,\ P \ni Y}{P \ni (X,Y),\ P \ni F(X,Y)}$$

P3 $$\frac{P \ni (X,Y)}{P \ni X}$$

P4 $$\frac{P \ni X}{P \ni H(X)}$$

P5 $$\frac{P \ni F(X,Y),\ P \ni X}{P \ni Y}$$

P6 $$\frac{P \ni K,\ P \ni X}{P \ni \{X\}_K,\ P \ni \{X\}_K^{-1}}$$

P7 $$\frac{P \ni +K,\ P \ni X}{P \ni \{X\}_{+K}}$$

P8 $$\frac{P \ni -K,\ P \ni X}{P \ni \{X\}_{-K}}$$

3.6.4 Freshness Rules

This set of rules help in determining a principal P's beliefs of freshness of a message as a result of a belief in the freshness of one of its components or freshness of a function of that message.

F1 $$\frac{P \models \sharp(X)}{P \models \sharp(X,Y),\ P \models \sharp(F(X))}$$

F2 $$\frac{P \models \sharp(X),\ P \ni K}{P \models \sharp(\{X\}_K),\ P \models \sharp(\{X\}_K^{-1})}$$

F3 $$\frac{P \models \sharp(X),\ P \ni +K}{P \models \sharp(\{X\}_{+K})}$$

F4 $$\frac{P \models \sharp(X),\ P \ni -K}{P \models \sharp(\{X\}_{-K})}$$

F5 $$\frac{P \models \sharp(+K)}{P \models \sharp(-K)}$$

F6 $$\frac{P \models \sharp(-K)}{P \models \sharp(+K)}$$

F7 $$\frac{P \models \phi(X),\ P \models \sharp(K),\ P \ni K}{P \models \sharp(\{X\}_K),\ P \models \sharp(\{X\}_K^{-1})}$$

F8 $$\frac{P \models \phi(X),\ P \models \sharp(+K),\ P \ni +K}{P \models \sharp(\{X\}_{+K})}$$

F9 $$\frac{P \models \phi(X),\ P \models \sharp(-K),\ P \ni -K}{P \models \sharp(\{X\}_{-K})}$$

F10 $$\frac{P \models \sharp(X),\ P \ni X}{P \models \sharp(H(X))}$$

F11 $$\frac{P \models \sharp(H(X)),\ P \ni H(X)}{P \models \sharp(X)}$$

3.6.5 Recognizability Rules

These rules help in determining whether a principal P recognizes a message as a result of recognizing a component, or a function of that message.

R1 $$\frac{P \models \phi(X)}{P \models \phi(X,Y),\ P \models \phi(F(X))}$$

R2 $$\frac{P \models \phi(X),\ P \ni K}{P \models \phi(\{X\}_K),\ P \models \phi(\{X\}_K^{-1})}$$

R3 $$\frac{P \models \phi(X),\ P \ni +K}{P \models \phi(\{X\}_{+K})}$$

R4 $$\frac{P \models \phi(X),\ P \ni -K}{P \models \phi(\{X\}_{-K})}$$

R5 $$\frac{P \models \phi(X),\ P \ni X}{P \models \phi(H(X))}$$

R6 $$\frac{P \ni H(X)}{P \models \phi(X)}$$

3.6.6 Message Interpretation Rules

These rules aid in determining new beliefs for a principal P, given an initial set of beliefs for P.

I1 $$\frac{P \triangleleft *\{X\}_K,\ P \ni K,\ P \models P \overset{K}{\leftrightarrow} Q,\ P \models \phi(X),\ P \models \sharp(X,K)}{P \models Q \sim X,\ P \models Q \sim \{X\}_K,\ P \models Q \ni K}$$

I2 $$\frac{P \triangleleft *\{X,< S >\}_{+K},\ P \ni (-K,S),\ P \models \overset{+K}{\mapsto} P,\ P \models P \overset{S}{\leftrightarrow} Q,\ P \models \phi(X,S),\ P \models \sharp}{P \models Q \sim (X,< S >),\ P \models Q \sim \{X,< S >\}_{+K},\ P \models Q \ni +K}$$

I3 $$\frac{P \triangleleft *H(X,< S >),\ P \ni (X,S),\ P \models P \overset{S}{\leftrightarrow} Q,\ P \models \sharp(X,S)}{P \models Q \sim (X,< S >),\ P \models Q \sim H(X,< S >)}$$

I4 $$\frac{P \triangleleft \{X\}_{-K},\ P \ni +K,\ P \models \overset{+K}{\mapsto} Q,\ P \models \phi(X)}{P \models Q \sim X,\ P \models Q \sim \{X\}_{-K}}$$

I5 $$\frac{P \triangleleft \{X\}_{-K},\ P \ni +K,\ P \models \overset{+K}{\mapsto} Q,\ P \models \phi(X),\ P \models \sharp(X,+K)}{P \models Q \ni (-K,X)}$$

I6 $$\frac{P \models Q \sim X,\ P \models \sharp(X)}{P \models Q \ni X}$$

I7 $$\frac{P \models Q \sim (X,Y)}{P \models Q \sim X}$$

3.6.7 Jurisdiction Rules

These rules help in determining conclusions derived by a principal P as a result of the belief of P in the authority of another principal Q.

J1 $$\frac{P \models Q \Rightarrow C,\ P \models Q \models C}{P \models C}$$

J2 $$\frac{P \models Q \Rightarrow Q \models *,\ P \models Q \sim (X \rightsquigarrow C),\ P \models \sharp(X)}{P \models Q \models C}$$

J3 $$\frac{P \models Q \Rightarrow Q \models *,\ P \models Q \models Q \models C}{P \models Q \models C}$$

3.6.8 Never-Originated-Here Messages

These rules result in the derivation of beliefs for a principal P about a component of a message which he/she has received and believes that it is not originated by himself/herself (denoted by P $\models \otimes$(P)).

I1′ $$\frac{P \triangleleft \{X\}_K,\ P \ni K,\ P \models P \overset{K}{\leftrightarrow} Q,\ P \models \phi(X),\ P \models \otimes(P)}{P \models Q \sim X,\ P \models Q \sim \{X\}_K}$$

I2′ $$\frac{P \triangleleft \{X, < S >\}_{+K},\ P \ni (S, -K),\ P \models \overset{+K}{\mapsto} P,\ P \models P \overset{S}{\leftrightarrow} Q,\ P \models \phi(X,S),\ P \models \otimes}{P \models Q \sim (X, < S >),\ P \models Q \sim \{X, < S >\}_{+K}}$$

I3′ $$\frac{P \triangleleft H(X, < S >),\ P \ni (X,S),\ P \models P \overset{S}{\leftrightarrow} Q,\ P \models \phi(X,S),\ P \models \otimes(P)}{P \models Q \sim (X, < S >),\ P \models Q \sim H(X, < S >)}$$

Chapter 4

Automated Protocol Analysis

Protocol analysis using formal analysis methods is often a tedious process to be done manually. For example methods using the Dolev-Yao model [6] use a system of communicating state machines and model an intruder who is in complete control of the system with the ability to read, alter and destroy messages, create new messages and perform legitimate operations such as encryption. Such systems involve considerable amount of data storage on the part of each of the principals and the intruder. Besides, storage space may also be required for messages in transit, either between each pair of principals or in general for every message sent and received. Additionally, many operations relating to communicating finite state machines are not well-suited for manual use. If the goal of the intruder in the model is to find out a secret word, this entails searching in a word space, which is infinite for protocols normally considered. All these suggest the development of systems for protocol analysis using machine assistance.

Systems based on modal logics consist of the systematic application of the inference rules of the logic to an idealized description of protocol messages. When applied for the first time, these logics may reveal missing assumptions

or redundancies in a protocol. The next step would be to revise the assumptions or modify the original protocol and reapply the inference rules to see if the desired goal of the protocol is then attainable. This process of applying and reapplying the inference rules is cumbersome and prone to errors, if done manually. Thus the need to automate the process was felt and a number of tools have been proposed to automate BAN logic [29, 12]. A tool to automate GNY logic is also developed [13] and will be discussed in section 3.1.2.

The importance of automating formal analysis of protocols is thus seen and we will describe some of the current work in this area and the need for further automation, in the rest of the sections of this chapter.

4.1 Current Developments

4.1.1 The Interrogator

The Interrogator [7] is a tool written in Prolog to detect vulnerabilities in a protocol. Given a protocol specification and a goal for the penetrator, it searches for a scenario using penetrator actions that achieves the desired goal. The highlight of the tool is that it prints a history of messages sent and modified which enables the user to actually view how the penetration was done and evaluate its feasibility and introduce possible measures to prevent such an occurrence. The tool performs an exhaustive search for penetrations within the allowed possibilities but avoids search paths that are evidently futile.

The Interrogator is one of the earliest tools to provide facilities to perform automated security analysis of protocols. The tool has a deep theoretical basis and is implemented in Prolog. It has a number of graphical user interface (GUI) features that enhance the automation process. This is described in more detail so as to offer a comparison with the GUI tool we propose and

describe in detail, later in this chapter.

The more salient features of the tool are explained below.

Communicating Finite State Machines

The Interrogator assumes each principal as a communicating finite state machine. The inputs and outputs of these machines are directed towards each other. Every message sent or received is an *event* and every event causes a *state transition.* Initially, each process or machine is assumed to be in a certain initial state. The various events occurring during the run of the protocol, i.e., the various messages sent and received in the protocol cause advancement from one state to another for all the affected principals (processes).

We assume an environment in which the intruder has complete control, i.e., he can inject new messages, and intercept or modify existing messages. Therefore sending and receiving a message are two separate events. A sent message may have been lost or a received message may have not been sent by a legitimate party and so on.

Transitions

Internally, the protocol description is converted and stored as a set of **transitions**. Transitions are specified as *transmit* or *receive* relations among process states or messages. The transmit relation defines state transitions caused by sending messages whereas the receive relation defines state transitions caused by receiving messages.

The transmit relation is specified as:

$$\text{transmit}(s_1, \text{m}, s_2)$$

which means that if a message m is sent by a process and s_1 is its state before the message is sent, then s_2 is its state after the message is sent.

The receive relation is similarly specified as:

$$\text{receive}(s_1, \text{m}, s_2)$$

where the message m sent by the sending process causes it to advance from a state s_1 to s_2.

State Structure

The essence of the system is the information carried by a protocol state. The following structure is used to represent the state attained by each principal (process):

$$[party, \text{state\_label}, item_1, ..., item_n]$$

The first element is the name or address of the principal and the second element is a state label. The rest of the elements are a list of data items that the process must retain. The state label is typically a mnemonic identifier such as OPEN or CLOSED or just an unique integer. When a message is sent or received the state label changes to the next one and some data items may be removed or more added. Sending a message may cause some items to be dropped and receiving a message may cause some items to be added to the list.

Message Structure

A message is represented as a list of fields. Each field is either a *simple* component or an *encrypted* component. An encrypted component is symbolically shown by prefixing the list of subfields being encrypted, by the key. The first two fields of the message are assumed to be source and destination addresses. An *address* is a party identifier type.

Thus message has the form :

$$[source, destination, field_1, ..., field_n]$$

An encrypted component has the form :

$$\text{key}[subfield_1, ..., subfield_m]$$

For example a message

$$\text{a} \rightarrow \text{b} : \{ck, \{ck, a\}_{kb}\}_{ka}$$

would be represented as

[a, b, ka[ck, kb[ck, a]]]

It is important to understand the difference between *variables* and *constants*. Each symbol represented in a protocol process state can be a variable or a constant. Variables occur in those places where the penetrator might cause variations. Constants occur in places where interfering with the symbol's value may cause the failure of the penetrator to achieve his goal. Changes in those places may cause such an abnormality that some principal may detect this and abort the connection and even inform other principals and abort the run of the protocol itself.

Apart from several communicating processes, we assume a separate *buffer* to hold messages in transit. Also it is assumed that the destination of any deliverable message can be determined from the content of the message. The *global state* of the network depends on the state of each of the communicating processes and the state of the network buffer. A *send* operation changes the state of the sending process and adds the sent message to the buffer. A *receive* operation changes the state of the receiving process and deletes the received message from the buffer. A *penetrator action* is any operation on the buffer which does not affect the state of any process. The penetrator can add a new message to the buffer, provided the buffer size bound is not exceeded and it can modify an existing message or delete an existing message in the buffer.

Penetration Analysis and User Interface

The version of **The Interrogator** which was studied, dealt with penetrator action to learn private information which is transmitted as data items in protocol messages. Simply stated, the Interrogator accepts a protocol specification and a target data item and it outputs a message history showing how the penetrator could target that item.

A *message history* is a sequence of message events consistent with the protocol specification. A *message event* is a term indicating transmission or reception of a message:

$$[\text{sent}(m_1), \text{rcvd}(m_1), ..., \text{sent}(m_n)]$$

Penetrator action can cause a sent message to disappear or be replaced by a different message. This will result in the *sent* and *rcvd* items not alternating with each other.

In the tool, vertical bars represent parties in the protocol and labeled arrows show the messages. Dashed lines show potential areas of interference by the penetrator. The figure shows an example of the following protocol, in which the penetrator interferes with message 1 and finally obtains the data item *data*1. This is a key distribution protocol in which *kdc* is a key distribution centre. It receives a request from party *a* to talk to party *b* and so generates a connection key *ck* and sends it to *a* and *b*, encrypted in their private keys *ka* and *kb*, respectively.

a $\rightarrow$ kdc : b

kdc $\rightarrow$ a : a, ka[ck]

kdc $\rightarrow$ b : b, kb[ck]

a $\rightarrow$ b : ck[data1]

b $\rightarrow$ a : ck[data2]

The *user interface* has two components: the preprocessor and the display

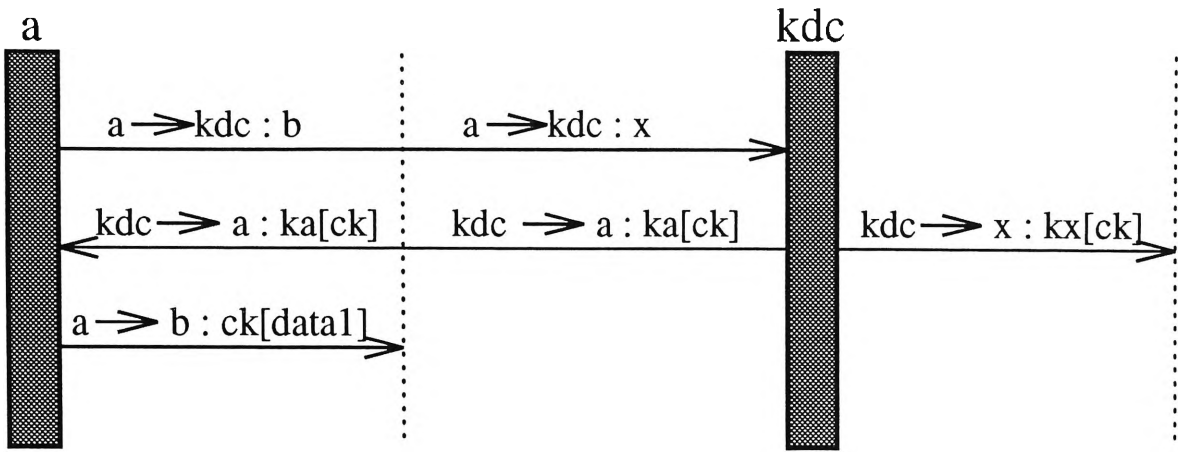


Figure 4.1: Sample Message History with Penetrator Actions

interface. The preprocessor accepts a protocol specification in a predefined format as described below and converts it into transmit and receive clauses for each communicating state process, as described previously. A sample protocol specification is given:

PROTOCOL example

CONSTANTS

a, b, x, kdc : address

ka, kb, kx, ck, oldck : key

d : data

RELATIONS

a, kdc : secret_key(a, ka)

b, kdc : secret_key(b, kb)

x, kdc : secret_key(x, kx)

KNOWNS

a, b, x, kx, oldck, kb[oldck, a]

MESSAGES

(* request ck *) [a, kdc, b]

(* generate ck *) [kdc, a, ka[ck, kb[ck]]]

(* forward ck *) [a, b, kb[ck]]

(* send data *) [b, a, ck[d]]

The CONSTANTS section lists the symbolic constants with their types, used in the protocol. The KNOWNS section lists all the constants and relations known to the penetrator before the protocol run starts. It also contains any information that the penetrator might have gathered from previous runs such as encrypted subfields or encryption keys. The RELATIONS section is used to denote any relationships assumed to hold between pairs of constants implying that if one value is known then the other can be calculated or looked up. Prefixing a relation with a list of addresses limits the relations as private to those addresses. In the above example, *secret_key* denotes a secret key relation of a principal with the key distribution centre, kdc.

In the conversion phase, the preprocessor takes the parse tree produced earlier and creates a state machine representation, one per principal in the protocol. Each protocol message is broken up into send and receive clauses.

The Interrogator can basically be characterized declaratively as a relation *p_knows*. This relation is:

p_knows(x, H, q)

It means that the penetrator can learn the value of the data item x at the conclusion of message history H, which takes the network from its initial state to q. This relation and the other ones described below are meant to reflect their implementation in Prolog. The basic definition of *p_knows* is as follows:

p_knows(x, H, q) iff

x is known initially

or (H = H'sent(m) and sent(m):q' $\rightarrow$ q

and H': $q_0 \rightarrow$ q' and p_gets(x, m, H', q'))

(H = H'e and e:q' $\rightarrow$ q and p_knows(x, H', q'))

or (H: $q_0 \rightarrow$ q' and p_modifies(q', q, H)

and p_knows(x, H, q'))

This means that there are three ways to deduce that the penetrator knows the item x by the time H reaches q: It can be known initially; it can be obtained from the last sent message; or it might have already have been known in the previous network state q'. This last case expands into two cases; one when the last state change was due to a message event and one when the change was due to a penetrator action.

The notation $H = H'e$ means event e is appended to message history H'. The notation $e : q' \rightarrow q$ denotes a network state transition from q' to q. *sent(m)* denotes the transmission of message m.

The penetrator is initially assumed to know public information like addresses of all parties in the network, and the private key of some subverted party or any other prespecified information. To extract a data item, for the penetrator, the following relation is used:

p_gets(x, m, H, q) iff

x is a field of m

or (k[m'] is a field of m and p_knows(k, H, q)

and p_gets(x, m', H, q)

This relation specifies that the penetrator can read any field of a message, but if the field is an encrypted submessage, the penetrator can extract a sub-

field from it only if the key encrypting that field is known. It is assumed that the context of the prior state and history is available for this purpose, as this is needed to support the recursive reference to *p_knows*, for the key.

Penetrator actions to modify the network buffer use the relation:

p_modifies(q', q, H)

It means that if m is a new message in the network buffer of the new state q, the penetrator knows each field of m in the prior state q', reached by history H.

Additionally, several computational operations on behalf of the penetrator are modeled, which are required to find attacks.

The Prolog implementation is built around these relations, to find attacks in a protocol. A suitable user interface helps helps the user in the choice of protocols, the target data item and a chosen final state, if any. The normal and modified message histories obtained are then displayed on the program's main window.

4.1.2 Automated Tool for GNY Logic

The aim of using modal logics like BAN and GNY for protocol analysis is their simplicity and effectiveness in analyzing protocols. The logics are systematically applied to protocols represented in a suitable idealized format as required by them This process can reveal missing assumptions or defects in the protocol. This would then warrant revising the protocol and/or the assumptions and reapplying the inference rules. This process of applying and reapplying inference rules of the logic is tedious and error-prone and has prompted many efforts to automate the reasoning process [12, 13].

GNY logic has more than forty inference rules and this increases the possibility of missing some necessary inferences. The large number of inference

rules also makes the reasoning process more laborious and as the GNY logic operates at a more finer level than BAN logic, proofs of goals attained tend to be much longer. All these problems have prompted the development of a tool for automating GNY logic, which we shall now describe.

The objective of this tool is to determine whether one or more statements defining the goal of the protocol are derivable from the given set of assumptions or not. The tool, in fact generates all statements that are derivable for a given protocol. Thus using this tool we can observe the states of all principals after each step in the protocol and also the final states achieved by all principals. The tool uses a forward chaining strategy to apply the logic in which the inference rules are repeatedly applied to the set of statements which consist of the initial assumptions, idealized protocol statements and derived statements until no further statements are derivable.

Many of the inference rules of the logic are not suitable for forward chaining, for example the freshness rule F1 [13]

$$F1 \quad \frac{P \models \sharp(X)}{P \models \sharp(X,Y)}$$

which states that the 'freshness' of a concatenated message can be derived from the 'freshness' of any of its concatenates. Since Y can be any formula, the above rule can be applied infinitely to derive the freshness of any formula containing X. Thus to automate the GNY logic analysis, the GNY postulate set was modified, i.e., some rules were modified, some inessential rules were deleted and some previously implicit rules were added.

Modification to the GNY postulates

In order to have finiteness of derivations, modifications to the postulate set were made. This entire set of modified rules is described in [13].

Implementation of the Tool

The tool provides a facility to represent logical constructs in an implementation language. It takes an input specification describing the idealized protocol and initial assumptions and outputs a complete set of logical statements derivable from the specification and the assumptions. It also provides a feature to extract proofs of statements derived. Proofs provide a mechanism to verify the protocol goals attained and also determine the role played by the various assumptions and protocol messages in the goal attainment. The tool is implemented in Prolog.

Protocol Specification

The various constants occurring in formulae within messages are represented by one or more lowercase letters. If K_{ab} is a session key for two principals A and B, it is written as a Prolog atom `kab`. The remaining functions like concatenation, encryption, functions etc, are represented as Prolog structures which closely resemble the standard structures in which they are typically represented. Extensions are specified using a construct `ext(X, C)` where `C` is the extension to a formula `X`. If there is no extension, `C` is specified as `nil`.

The following tables illustrate the equivalent structures for formulae and statements used in GNY logic :

Formula	Structure
(X,Y)	`[X, Y]`
$\{X\}_K$	`encrypt(X, shared(K))`
$\{X\}_K^{-1}$	`decrypt(X, shared(K))`
$\{X\}_{+K}$	`encrypt(X, public(K))`
$\{X\}_{-K}$	`decrypt(X, private(K))`
$H(X)$	`h(X)`
$F(X_1,.....,X_n)$	`f(X1, ...., Xn)`
$*X$	`star(X)`
$X \rightsquigarrow C$	`ext(X, C)`
X	`ext(X, nil)`

Statement	Structure
$P \triangleleft X$	`told(P, X)`
$P \ni X$	`possesses(P, X)`
$P \mid\sim X$	`conveyed(P, X)`
$P \models \sharp(X)$	`believes(P, fresh(X))`
$P \models \phi(X)$	`believes(P, recognizes(X))`
$P \models Q \overset{S}{\leftrightarrow} R$	`believes(P, secret(Q, S, R))`
$P \models \overset{+K}{\mapsto} Q$	`believes(P, public(K, Q))`
$P \models C$	`believes(P, C)`
$P \models Q \Rightarrow C$	`believes(P, controls(Q, C))`
$P \models Q \Rightarrow Q \models *$	`believes(P, honest(Q))`
C_1, C_2	`[C1, C2]`

For example the idealized statement

$$A \triangleleft *\{N_a, B, *K_{ab}\}_{K_{as}} \rightsquigarrow S \models A \overset{K_{ab}}{\leftrightarrow} B$$

is written as

```
told(a, ext(star(encrypt([na, b, star(kab)], kas), believes(s, secret(a,
kab, b)))))
```

Derived Statements

In order to represent derivation information of statements obtained by applying inference rules, a Prolog predicate `fact/3` which defines an inference step, is used. Its format is

```
fact(Index, Stat, reason(PremIs, Rule))
```

`Index` is an integer argument used to index instances of `fact/3`. `Stat` is the actual derived statement and `PremIs` is a list containing indices of premises used to derive `Stat` by applying the rule `Rule`. `fact/3` is also used to describe idealized statements and assumptions to maintain consistency. `PremIs` is empty if `Stat` is either an inference step or an assumption and then `Rule` is appropriately `'Step'` or `'Assumption'`.

Usage of the Tool

The idealized description of the protocol to be analyzed is converted into Prolog syntax using the conversion tables shown previously. The set of facts denoting the idealized protocol description and initial assumptions is loaded into the analyzer to derive all logical statements derivable. This is done as:

```
| ?- analyze(<spec-file>).
```

After this the facts so generated are available to be queried for protocol

goals attained. The predicate `fact/3` is used with a dummy index `I` and `Rule` as follows

```
| ?- fact(I, Stat, Rule).
```

where `Stat` is the desired goal. If the goal is indeed attained the analyzer outputs the index of the stored fact as `I` and `Rule` which contains the `PremIs` and actual rule resulting in `Stat` being attained.

explain_proof is used to obtain proofs of derived statements :

```
| ?- explain_proof(Stat).
```

where `Stat` is an already derived statement. The steps leading to the derived statement are then listed. Each step is either an assumption or a conclusion drawn from earlier steps. The line numbers of premises used in the derivation and the inference rule are also output to the right of each statement.

4.2 Need for Further Automation

The automated tool for GNY logic described in the previous section is useful but still is difficult to use for the new user. Since the tool is implemented in Prolog, all the logical constructs for formulae and statements are Prolog structures. The idealized description and initial assumptions are also represented using the Prolog predicate `fact/3` which is used to typify inference steps within the analyzer.

These requirements impose a need for the user of the tool to have elementary knowledge about Prolog and its syntax. The version of Prolog in which the tool has been implemented (XSB Version 1.4.0) operates in an environment of its own. Thus basic Prolog startup commands have to be known.

The correspondence between the Prolog structures used in the syntax and standard structures used in protocol messages is not easily apparent. It is

especially difficult for a new user of the tool who is totally unfamiliar with the Prolog environment, to try to convert the idealized protocol specification and assumptions to prolog syntax and input them to the tool, without the possibility of errors. A critical point is that the protocol messages input to the tool have to be in idealized form as required for the application of GNY logic. This imposes an added task of transforming standard protocol messages to include *not-originated-here* markers (*'s) and extensions to message components. This is a sufficiently cumbersome process, not to be done manually as the equivalent of *GNY parsing* has to be done on messages to insert the markers. A similar but slightly simpler effort is needed to append extensions to messages.

Another aspect is that while using GNY logic for protocol analysis, the sets of beliefs attained by all the principals keep increasing in size and this increase is evident if the analysis is performed step by step. Due to the large number of inference rules in the logic and detailed process of applying them, the beliefs increase in length and number for each principal and it is difficult for a user to keep track of the beliefs of each principal, particularly if he wishes to step through the analysis of the protocol for each message. This aspect is also true for proofs of attained goals as proofs can become lengthy and unweildy.

The aim is to provide a graphical user interface to this tool and provide features that attempt to minimise the problems in using the automated GNY tool. Basically our system has two main objectives:

1. **Computer aided learning tool for protocols**

   Many protocols are designed nowadays for authentication and key exchange purposes. These are described sometimes using standard notations or notations defined for the purpose. The goals of these protocols, the roles of various principals and the flow of messages between them is not readily discerned by reading the standard protocol description. Our

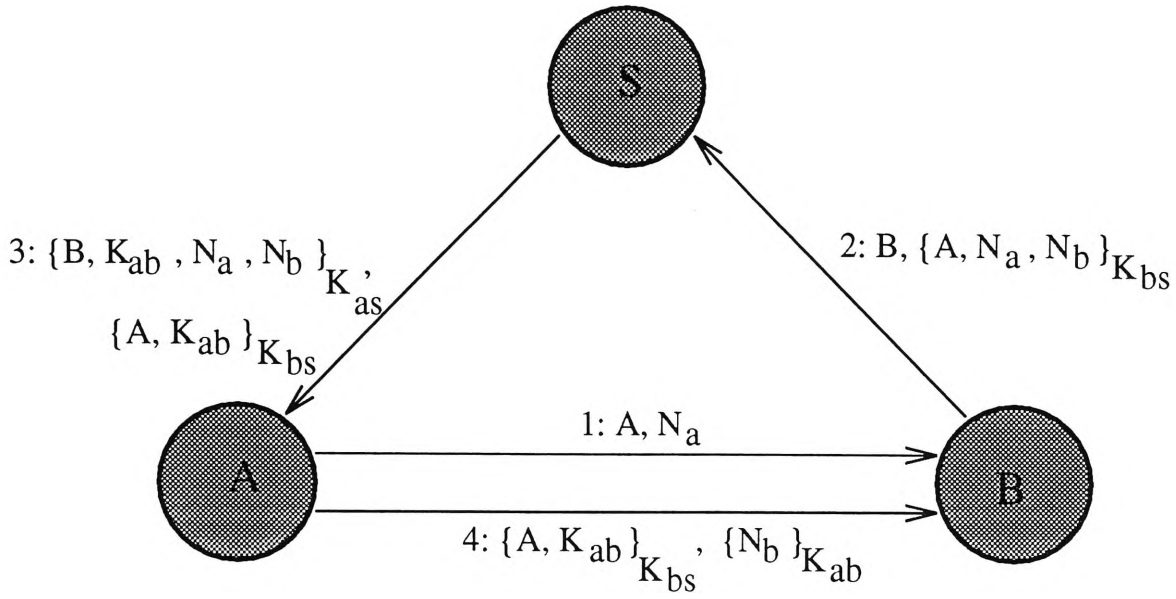


Figure 4.2: Schematic Diagram of Yahalom Protocol

system is expected to provide a user-friendly environment for displaying protocol runs. As seen in literature [4], protocols are often described using schematic diagrams and this can greatly enhance their understanding, particularly in relation to the above mentioned aspects in protocols. Such a diagram would show the principals as circular nodes and potential channels of communication between principals as arcs between them. A single run of the protocol would then be shown as a sequence of messages, each of which is shown to flow from the sender to the receiver using some distinguishable graphical techniques. Also, the message text could be displayed alongside the arc. It would be useful to single-step through the protocol and this could facilitate better understanding of the positions of various principals at each step in the protocol.

A schematic diagram of the Yahalom protocol [4] is shown to illustrate the standard form of diagrammatic representation, in literature.

2. **Further Automation**

The system we describe could fulfill the need for a top layer interface to the automated GNY tool. Since our system would provide a syntax of its own for the input of the protocol specification, which is not required to be in the idealized form required by GNY logic, this is expected to alleviate the need to use Prolog structures and human reasoning to insert $not - originated - here$ markers and extensions to messages. Also this syntax enforces some other conditions which are implicit in an idealized protocol specification.

On the other hand, the graphical user interface takes care of interaction with the GNY tool and provides an environment to show protocol sessions and also displays the beliefs of all principals during the course of protocol execution and also finally. Also the ability to extract proofs of any derived statement graphically is expected to build on the automation provided by the GNY tool and enhance its usage and overcome some of its problems and thereby provide more ease and assistance in the analysis of many protocols.

4.3 Comparison with 'The Interrogator'

We detail some of the comparisons of facilities provided by our proposed protocol analyzer and its user interface with those provided by the Interrogator.

1. More information has to be provided to the Interrogator compared to that for our tool. For example, the constants (data items), the precise relations between the protocol entities e.g. keys etc and information known to the penetrator have to be specified as input. This compares with our tool where we propose to take the concrete messages only along with initial assumptions.

2. The Interrogator specifies only protocols that are parsed and translated into prolog syntax can be selected for use within the tool whereas in our tool we can select any protocol from the family of protocols handled by GNY logic, and those that can be specified in the syntax of the analyzer.

3. The Interrogator concentrates on a penetration objective, particularly accessing cleartext data items within a protocol. In our tool, due to the step-wise depiction and display of beliefs, many different conclusions can be drawn about the state of the principals as also the final states of all the principals.

4. The Interrogator displays the entire protocol run with the message sequence, but in our tool besides having this facility we propose to have a mode for step-wise display of protocols messages and beliefs. We also propose to display proofs of any derived belief at any stage in the protocol.

5. We believe that the diagrammatic representation that we propose, that is, the use of circular nodes for principals and messages displayed as arcs between them, aids the understanding of the protocol better than the user interface used by the Interrogator.

4.4 Conclusion

We have seen the need and advantages of automating the process of formal analysis of protocols and how the use of such automated tools can greatly assist in the process and finally be an aid in the design of optimal protocols. The next chapter will describe in detail the design and implementation of the protocol analyzer.

Chapter 5

The Protocol Analyzer and the User Interface

As described in the previous chapter, the system we are proposing serves the purpose of a computer aided learning tool and a top-layer graphical user interface for a tool that implements logical analysis of cryptographic protocols. In this chapter we will discuss the features of the proposed system and give an overview of its design.

5.1 The Protocol Analyzer

The user interface system which is proposed will hereinafter be referred as the *Protocol Analyzer*. As outlined in the previous chapter, it has two main goals:

1. A computer based educational tool for protocols
2. A front-end user interface for the automated GNY tool.

As a computer based educational tool, the *Protocol Analyzer* provides the following features:

1. Display of the chosen protocol in the form of a schematic diagram.

2. Facility to show the running of the protocol using the block schematic representation.

3. Facility to show single stepping of the protocol, using the block schematic representation and other features outlined below.

As part of the objective of providing a top layer interface to the automated GNY tool, the following features are provided:

1. An independent language for specification of protocol messages.

2. A translator for translating messages specified in this language to idealized form in Prolog.

3. Facility to display beliefs or derived statements and initial assumptions, initially and during the execution of the protocol.

4. Facility to display proofs of beliefs obtained, during the execution of the protocol.

5. Facility to modify the initial assumptions and rerun the protocol, during a session of a protocol.

The representation of a protocol in the form of a schematic diagram is expected to enhance its understanding from that gained by merely going through its concrete specification. The single stepping feature provided is expected to provide further insight into the execution of protocols and the roles of various principals in it. This feature combined with the display of beliefs of principals and proofs of derived beliefs is provided as an easier interface to the automated GNY tool. The protocol specification language provided is expected to help in specifying protocols, easily as compared to specifying protocols in idealized

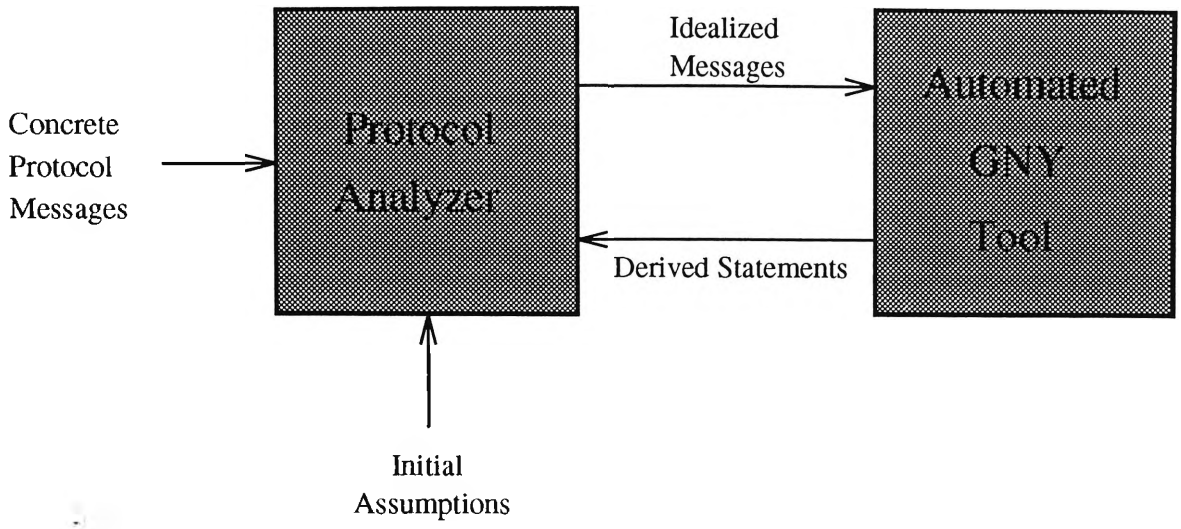


Figure 5.1: The Function of the Protocol Analyzer

format in Prolog. The translator translates messages specified in the syntax of this language into the idealized form.

The function of the *Protocol Analyzer* is shown diagrammatically, in the figure. The various inputs and outputs from the *Protocol Analyzer*, and the interface between it and the automated GNY tool are also shown. The features outlined above are described in further detail in the next section.

5.2 Features

5.2.1 Syntax for Protocol Specification

As mentioned in previous chapters, one of the shortcomings in using the automated GNY tool directly is that it requires the protocol specification in the idealized form as required by GNY logic [5]. The specification is also required to be in Prolog syntax, using Prolog constructs which are defined. These were listed in the two tables in the previous chapter.

As far as specifying protocol messages is concerned, the Prolog structures defined for the automated GNY tool are made to resemble their concrete coun-

terparts as closely as possible. Our effort has been to bridge this gap and make message components look similar to how they are described, in literature. It must be understood however, that structures for encrypted, decrypted and hashed function components and components involving groups of subscripted variables cannot be represented as they are in a textual form of input. This could be done if input is accepted in graphical form using a graphical text editor, but this feature is not currently implemented and could be a useful extension to the existing system.

The concrete protocol messages are specified in a single file whose name is supplied as an argument to the *Protocol Analyzer* main program. The syntax followed for specifying messages is given in the table below. The formulae listed in the table conform to the standard notation used to describe message components. For each formula is given the corresponding construct to be used for the *Protocol Analyzer*. These constructs are then used to form messages in the file to be given as input to it.

Formula	Structure
(X,Y)	`[x,y]`
$\{X\}_k$	`enc(shared(k)){x}`
$\{X\}_k^{-1}$	`dec(shared(k)){x}`
$\{X\}_{+k}$	`pub(public(k)){x}`
$\{X\}_{-k}$	`pri(private(k)){x}`
H(X)	`h{x}`
F(X_1, X_2, ..., X_n)	`f(x1, x2, ..., xn)`

It is assumed that all information in the specification will be in lowercase.

This is done to maintain uniformity as the GNY tool requires input to be in lowercase and outputs from the tool, i.e., beliefs and proofs obtained will conform in style with the input description, while using names of principals, keys etc.

Names of keys or other subscripted symbols are specified by appending the subscripts to the symbol names in lowercase. In a message line, a comma and a single space are assumed as delimiters between two different components. Each message is written on a single line starting with the number of the message followed by a ')' and a space. This is followed by the name of the sender of the message. Then the characters '$- >$' are written. This is meant to resemble the arrow showing the flow of the message from the sender to the receiver in concrete protocol descriptions. After these characters the name of the receiver is written followed by a ':'. All the characters occurring after this delimiter will be treated as message text by the text parser and translator. The text may contain names of principals involved in the protocol and other components conforming to the syntax given in the table, previously. Keys occurring within the message text need to be declared as *shared*, *private*, or *public* depending on the cryptosystem used.

Comments in protocol specification files are specified by placing a '%' character in the first column of the line. When all messages are built in the manner indicated above, the file can be given as input to the *Protocol Analyzer*.

We give an example of the Otway-Rees protocol [2] to show how it can be represented in the syntax described above.

Otway-Rees Protocol

1. A $\rightarrow$ B : M, A, B, $\{N_a, M, A, B\}_{K_{as}}$

2. B $\rightarrow$ S : M, A, B, $\{N_a, M, A, B\}_{K_{as}}$, $\{N_b, M, A, B\}_{K_{bs}}$

3. S $\rightarrow$ B : M, $\{N_a, K_{ab}\}_{K_{as}}$, $\{N_b, K_{ab}\}_{K_{bs}}$

4. B $\rightarrow$ A : M, $\{N_a, K_{ab}\}_{K_{as}}$

This description is translated using the table given as:

```
1) a -> b :  m, a, b, enc(shared(kas)){[na, m, a, b]}
2) b -> s :  m, a, b, enc(shared(kas)){[na, m, a, b]},
             enc(shared(kbs)){[nb, m, a, b]}
3) s -> b :  m, enc(shared(kas)){[na, shared(kab)]},
             enc(shared(kbs)){[nb, shared(kab)]}
4) b -> a :  m, enc(shared(kas)){[na, shared(kab)]}
```

Limitations

A limitation of the syntax described above is that protocol messages consisting of nested components i.e. encryption within encryption, hashing within hashing and so on cannot be specified in the syntax. Hence protocols consisting of such messages cannot be analyzed using the *Protocol Analyzer*, in the current implementation. This is due to constraints on implementation of the text translator and can be a good enhancement to the system in future.

Some other limitations of the *Protocol Analyzer* and associated further work are discussed at length in the concluding chapter.

Initial Assumptions

All the principals in the protocol are assumed to have a set of initial beliefs before the start of the protocol, in the application of GNY logic. This initial set of beliefs and assumptions are also required to be input in the form of a

text file to the *Protocol Analyzer*. The *Protocol Analyzer* will later use this as input to the automated GNY tool, when it is invoked.

The syntax used for specifying initial assumptions is the same as that used for input to the GNY tool. This is done to avoid complexity in implementation. The assumptions are specified using the Prolog predicate `fact/3`. The index of facts, i.e., the first argument to the predicate must begin from one after the number of messages listed in the protocol specification file. When the initial assumptions of all principals are listed, a predicate `flag` is used to declare the index of the last fact instance to the GNY tool. The file containing the initial assumptions can then be supplied as input to the *Protocol Analyzer*.

5.2.2 Messages Translator

The main task of the messages translator is to transform the protocol messages from the syntax defined for the user to the idealized form required by GNY logic. All the messages are translated into idealized form and stored internally. The idealized messages and initial assumptions are later used by the *Protocol Analyzer* as input to the GNY tool. The actual design of the messages translator will be discussed in later sections.

5.2.3 Diagrammatic Representation of Protocols

As described previously, the form of representation used for schematic display of protocols in the *Protocol Analyzer* is similar to that used in literature [4, 13]. We start by describing the look and feel of the graphical user interface.

The *Protocol Analyzer* has a main window which consists the schematic diagram of the principals of the protocol and other control panes. The control panes are four mouse-sensitive areas or buttons which occur in the top of the window. These buttons are *Go*, *Run*, *Re-edit* and *Quit*. The functions of these

buttons are described below:

GO : This button is used to activate the *single-step* mode of the *Protocol Analyzer*. This is also the mode in which the it interacts with the GNY tool and displays the beliefs in each principal's subwindow after each step. The *single-step* mode is explained in detail later.

RUN : This button is used to activate the *full run* of the protocol. On activating this button, the full sequence of messages is run through by showing the flow from sender to receiver for each message. The *full-run* mode is also explained later.

RE-EDIT : This button is used to activate a text editor from within the *Protocol Analyzer* to edit the initial assumptions of the protocol principals. After the changes are made and the *Go* button is activated the single stepping of the protocol begins again. The *Re-edit* feature is explained in a later section, in detail.

QUIT : The quit button is used to exit the *Protocol Analyzer*.

In the schematic diagram, all the principals in the protocol are shown as circular objects. These circular objects are connected by arcs, representing communication channels between the principals. A circular object contains the name of the principal within itself, as specified in the protocol specification file.

Status Line

The lowermost portion of the main window displays a *status line.* This is a single line of text which is drawn in reverse-video with a default message displayed within it. The status line is used to display diagnostic messages during the various modes and when quitting the *Protocol Analyzer.* This is especially useful when single-stepping through the protocol with beliefs being displayed, as the status line displays the mode and current message number being stepped through.

5.2.4 Modes of the *Protocol Analyzer*

The *Protocol Analyzer*, initially is in a default wait state, waiting for input from the user. The user can enter one of the two modes: *Full-Run* mode or *Single-Step* mode for viewing the execution of the protocol. The details of these modes are described here.

Full-Run mode

In this mode, the full execution of the protocol is shown. Once the *Protocol Analyzer* is invoked from the command line, the main window comes up, and the schematic diagram of the protocol specified on the command line is drawn on it. The system then enters a normal *wait* mode, waiting for input from the user.

As explained before, the user may invoke the *full-run* mode of the protocol by clicking the mouse on the *Run* button. On clicking this button, the first message of the protocol is displayed along the arc joining the sender and the receiver principals of the message. The message text is displayed in the syntax defined for the protocol input specification. The actual flow of the message is shown when the arc is highlighted by a thick line in reverse-video, terminated

by an arrow to show the direction of the message from the sender to the receiver. The status line is updated to reflect step 1. After a brief delay, the next message transition is displayed in the same manner.

The process outlined above is repeated for all messages in the protocol. After the last message is displayed the main window and the protocol schematic diagram are redrawn as they were initially and the system returns to its *wait* state.

5.2.5 Single-Step Mode

From the normal *wait* state of the *Protocol Analyzer*, the single stepping mode can be activated by clicking the *Go* button.

In the single-step mode, as opposed to the *Full-run* mode, the user can intervene after each protocol step is over.

The main window contains a number of subwindows [1] for displaying beliefs. There is a beliefs subwindow for each principal in the protocol which will be used to show its beliefs as the protocol progresses. Initially each belief subwindow displays the initial assumptions.

When the user clicks on the *Go* button, the *single-step* mode is activated. In this mode, each message exchange is shown in the same way as in the *full-run* mode. In addition, beliefs of the sender and the receiver are also updated. The *Protocol Analyzer* appropriately interacts with the GNY tool for this purpose. To see the next message transition and the next set of beliefs the user has to click on the *Go* button again and the same process repeats. This can be done for all the messages in the protocol. No other option except *Quit* and *Re-edit* can be selected until all the messages are stepped through.

[1] These are actually windows which are children of the root display window in Xlib. We refer to them as subwindows in order to differentiate from the main window.

5.2.6 Displaying Beliefs

In the graphical user interface of the *Protocol analyzer*, a subwindow is created for displaying the beliefs for each principal. Initially in the *Protocol Analyzer*'s wait state, these subwindows display the initial assumptions.

The beliefs subwindows are *scrollable*, meaning that if the number of beliefs exceed the number of rows that can be displayed entirely in a subwindow the user can view the undisplayed beliefs by scrolling forward on the scrollbar. A mouse-sensitive scrollbar is present in the left of every beliefs or proofs subwindow. Text can be scrolled forwards and backwards. The beliefs text pane is also mouse sensitive and when the user clicks on any belief a proofs subwindow comes up which displays the proof leading to that derived statement.

Within the subwindow, the beliefs are wrapped, meaning beliefs longer than the length of the subwindow are continued to the next line. Each belief subwindow displays a title showing which principal's beliefs it is actually displaying.

5.2.7 Proofs

Proofs provide a means to verify the attainment of a derived statement during the course of the protocol run or after it is over. The *Protocol Analyzer* provides a facility to display proofs of attained beliefs within the graphical user interface.

In the *single-step* mode, the user can view the proof of any derived belief which is currently displayed in a beliefs subwindow. When the mouse is positioned over the belief for which the proof is required and clicked, a proof subwindow similar to the beliefs subwindow comes up and it displays the set of statements leading to the belief. The proof is displayed exactly as obtained from the GNY tool, in the same syntax. A single proof subwindow is shown

in the figure. The proof subwindow is also scrollable forwards and backwards and proofs are continued on following lines within the subwindow.

5.2.8 Re-edit Facility

In the main window, the *Re-edit* button activates a text editor to edit the initial assumptions.

When the user is running the protocol in *single-step* mode, he may at any time click on the *Re-edit* button. This will cause spawning of a text editor with a copy of the initial assumptions file loaded for editing. The actual editor spawned depends on the value of the environment variable **EDITOR**, which is expected to be set. If this variable is not set, the default editor *vi* is used. When the user completes the changes to the initial assumptions, he can save the file and quit the editor. The window with the text editor will then disappear and a new set of belief subwindows will be displayed. A beliefs subwindow will be created for each principal and the modified initial assumptions will be displayed on it. The previous set of belief subwindows are left as they are.

From this point on, the user can click on *Go*, and *single-step* through the protocol with the new set of initial assumptions. When the single-stepping reaches the same step, at which the re-editing was activated, the new beliefs subwindow would display a set of beliefs which would possibly be different from the previous set. This would enable a comparison between the previous set and the new set of beliefs and can be used to study the effect of modification of initial assumptions on the statements derived at the end of that step and other steps and also at the end of the protocol session.

It is important to note that the re-editing process can be performed only once and further changes to the initial assumptions can only be made from outside the *Protocol Analyzer* environment and the it must be run again with

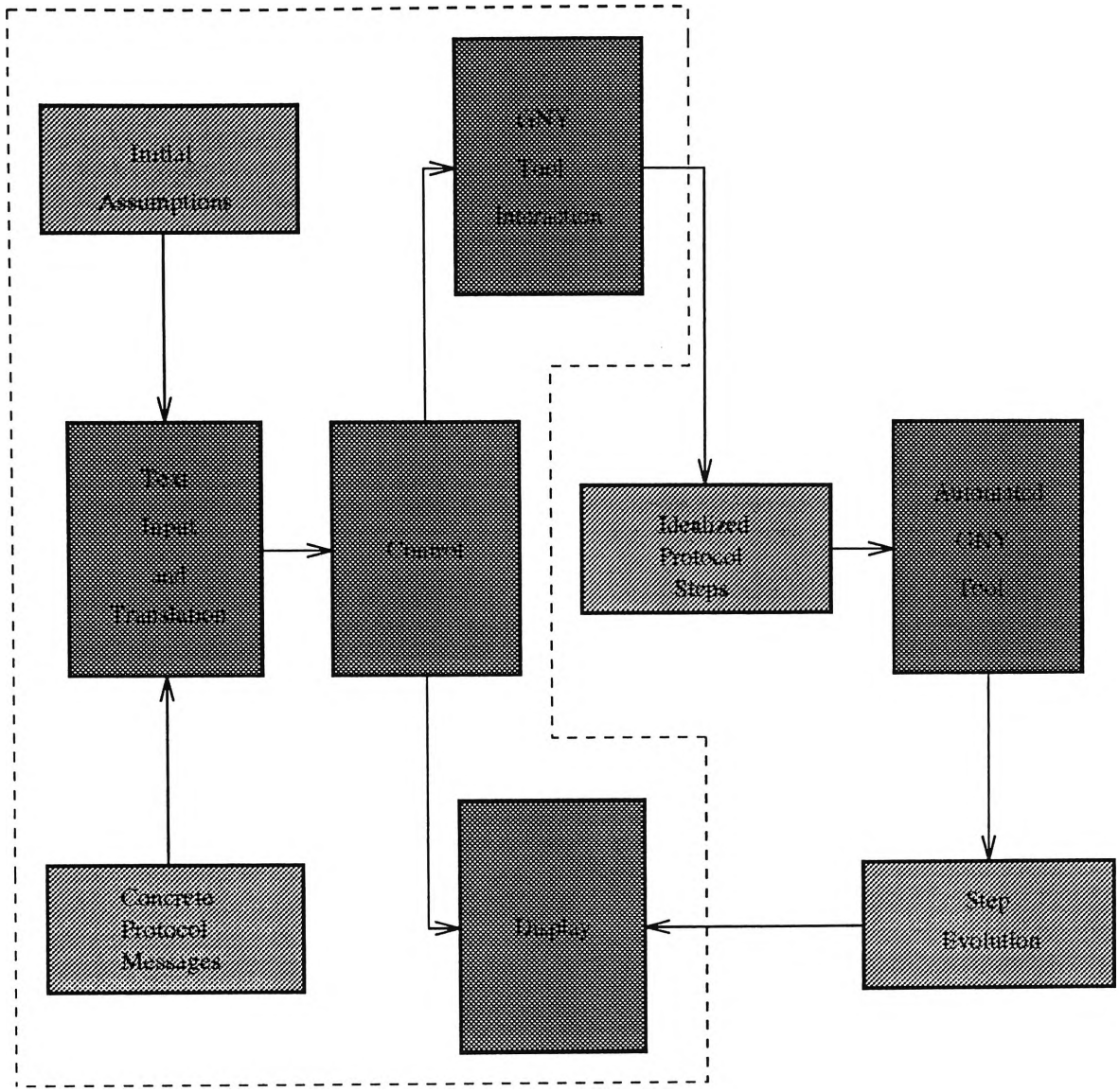


Figure 5.2: Protocol Analyzer - Block Schematic Diagram

the modified assumptions file, from the command line.

5.3 Program Components and Design

Figure 5.2 is a block diagram of the *Protocol Analyzer*. The portion of the diagram enclosed in dashed lines shows the implementation of the computer aided learning tool and the user interface for the automated GNY tool.

Based on the diagram, the following tasks for the *Protocol Analyzer* can be

identified:

1. Control
2. Text Input and Translation
3. GNY tool interaction
4. Display

5.3.1 Control

Control is the main controlling module of the system. It interacts with the text input modules to transform the protocol messages into idealized form. It is also responsible for creating and maintaining the windows and subwindows of the *Protocol Analyzer*. It also interacts with the *GNY* modules to obtain principals' beliefs at each step of the protocol and controls all these operations whilst waiting for input on the *Protocol Analyzer* main window.

The control module due to the nature of its tasks is not built as a class. The detailed source code appears in Appendix A.

5.3.2 Text Input and Translation

The text input and translation module performs text input and translation to the idealized form required by GNY logic. The concrete protocol messages and initial assumptions are read from input files and the messages are translated into idealized form for subsequent input to the GNY tool.

5.3.3 GNY Tool Interaction

This module interacts with the automated GNY tool during protocol analysis. It provides the idealized messages and principals' beliefs to the GNY tool and

the GNY tool returns the modified set of beliefs derivable as a result of the idealized messages provided to it.

In a similar manner, this module provides a belief statement together with a set of initial assumptions and an idealized protocol to the GNY tool, and it returns a proof of that belief (if it is provable).

5.3.4 Display

The *Display* module performs all the graphical user interface operations like drawing various components, drawing the protocol schematic diagram, display of beliefs and proofs in subwindows and handling graphical attributes, like scrolling text and handling mouse clicks on these windows.

5.3.5 Flow of Information

In the *Protocol Analyzer*, figure 5.2, we can observe how information flows from input to output.

The *Protocol Analyzer* accepts the initial assumptions and concrete protocol messages of the protocol as input. These are supplied to the *Text Input and Translation* module. This module validates the input specified in the language of the *Protocol Analyzer* and translates it into the idealized form as required by the automated GNY tool.

The initial assumptions and idealized messages are sent to the *GNY Tool Interaction* module by the *Control* Module. The *Control* module also supplies the initial assumptions and the concrete protocol messages to the *Display* module for display on the program's windows. The *GNY Tool Interaction* module provides the current set of beliefs (which are the initial assumptions at the start of the protocol) and the idealized messages to the automated GNY tool, which returns the updated set of beliefs for all principals in the protocol.

This is then sent by the *GNY Tool Interaction* module to the *Control* module which displays them by providing them to the *Display* module.

The *Control* phase coordinates this flow of information at all times.

The entire coding for the system is done using SC3.0.1 C++ [30] and X11R5/Xlib [14, 15] for graphical support. The modules associated with the above tasks are organized into classes except for *Control.* Instanced objects of these classes are used in the implementation. All the header and source files for all modules' appear in Appendix A.

Chapter 6

Results of Protocol Analysis

The main features of the *Protocol Analyzer* were described in the previous chapter along with an overview of its design. Examples of application of the *Protocol Analyzer* in the analysis of authentication and key-exchange protocols are included in this chapter.

6.1 Analysis of known Protocols

The *Protocol Analyzer* is used to examine three well known protocols:

1. Voting Protocol
2. Otway-Rees Protocol

For each protocol, the input protocol specification is written according to the syntax described in chapter 5. This specification together with the initial assumptions are then used as input to the *Protocol Analyzer*. For each protocol, beliefs of each principal were displayed after every step.

Proofs of certain beliefs, displayed by the *Protocol Analyzer*, were examined and the effect of changing initial assumptions on principals' beliefs were studied.

In the remaining sections of this chapter, the highlights of these results are presented for the Voting and Otway-Rees protocols.

6.2 Voting Protocol

The Voting Protocol is described in literature [4, 5]:

1. $Q \rightarrow P_i : N_q$
2. $P_i \rightarrow Q : P_i, N_i, V_i, H(N_q, < S_i >, V_i)$
3. $Q \rightarrow P_i : R, H(N_i, < S_i >, R)$

where P_i is a participant in the voting and Q is the coordinator. V_i is the actual vote sent by P_i to Q and R is the result announced by Q. N_q and N_i are nonces generated by Q and P_i respectively and $H()$ is a one-way hash function.

This protocol specification in the syntax of the *Protocol Analyzer* is:

```
1) q -> pi : nq
2) pi -> q : pi, ni, vi, h{[nq, si, vi]}
3) q -> pi : r, h{[ni, si, r]}
```

The above protocol specification is written using the syntax described in section 5.2.1. The keywords that can be used in the syntax are given in table 1 in section 5.2.1.

The initial assumptions for principals P_i and Q are described in Prolog syntax as:

```
fact(4, possesses(pi, si), reason([], 'Assumption')).
fact(5, possesses(pi, ni), reason([], 'Assumption')).
fact(6, believes(pi, secret(q, si, pi)), reason([], 'Assumption')).
```

```
fact(8, possesses(q, si), reason([], 'Assumption')).
fact(9, possesses(q, nq), reason([], 'Assumption')).
fact(10, believes(q, secret(q, si, pi)), reason([], 'Assumption')).
fact(11, believes(q, fresh(nq)), reason([], 'Assumption')).
flag(count, 11).
```

These initial assumptions are written as *facts* in idealized form. These are directly loaded into the automated GNY tool *facts* database. Each *fact* specified above is written using the syntax described in section 4.1.2. Each *fact* above describes a belief held by a principal before the start of the protocol using keywords given in tables 1 and 2 in section 4.1.2. For initial assumptions as above, the reason specified is always *Assumption.* The keyword `count` used in the last *fact* above is a keyword to denote that the number following it is the number of facts specified in the initial assumptions.

6.2.1 Results

On providing the *Protocol Analyzer* with the above input, a schematic diagram of the Voting Protocol is displayed. This is shown in the figure 1 in the appendix to this chapter.

In this state, the full-run mode is invoked in which case, every step of the protocol is shown by an arrow from the sender to the receiver accompanied by the message text. The beliefs subwindows are not updated in this mode. This mode gives us a quick understanding of how messages are actually transmitted in the Voting protocol.

A beliefs subwindow for principal q is shown in figure 10 in the appendix, under section B.1.2. It lists the beliefs attained by q after step 2 in the protocol.

By clicking on the belief 2, we obtain a proof for it in a separate proofs subwindow which is figure 3 in the appendix.

In this case we wish to see how the belief `told(q, pi)` was derived. The proofs subwindow shows 2 steps for this belief. In the first step, q attains the belief:

`told(q, [pi, ni, vi, ext(star(h([nq, si, vi])), possesses(pi, si), possesses(pi, ni), believes(pi, secret(q, si, pi)))])`

because of the protocol step 2. Note that the above is the idealized form of step 2.

As a result of this step and *Being-told rule 2*, the belief `told(q, pi)` is derived.

Modification of initial assumptions

After stepping through the Voting protocol, we would like to study the effect of changing initial assumptions on the final beliefs attained by pi and q.

We delete assumptions corresponding to facts with numbers 4 and 6, shown above. These mean that pi no longer possesses si and it does not believe that si is a secret between itself and q, before the start of the protocol.

When the protocol is re-run and the final beliefs attained by pi and q are displayed, we observe that, principal q has attained new beliefs all implying that nq, si and vi are *conveyed*. This means that q believes that nq, si and vi were *told* to it by pi but not necessarily in the current run. On the other hand pi believes that nq, si and vi when it *conveyed* them.

Thus it is seen that some different conclusions have been reached by q as a result of modifying the initial assumptions of pi. The goal of the protocol was that q believes that nq, si and vi were told to it by pi in the current run, which could be attained with the original set of assumptions. With the modified set, the beliefs representing the goal of the protocol cannot be attained.

On the other hand, if we only suppress fact 6 and re-run the analysis, the above belief is still attained implying that the initial belief represented in fact

6 is redundant. Repeating similar experiments for other assumptions, we can conclude that some of the assumptions in the original set are redundant.

Thus the above experiment shows that the *Protocol Analyzer* is able to assist in the protocol analysis process by identifying important initial assumptions and the effect of changing them on the final outcome of the protocol.

The windows showing q's final beliefs before and after the change in assumptions are given in figures in Appendix B, section 3.

Other detailed results obtained for the analysis of Voting protocol are given in Appendix B.

6.3 Otway-Rees Protocol

The protocol description for Otway-Rees protocol is:

1. $A \rightarrow B : M, A, B, \{N_a, M, A, B\}_{K_{as}}$
2. $B \rightarrow S : M, A, B, \{N_a, M, A, B\}_{K_{as}}, \{N_b, M, A, B\}_{K_{bs}}$
3. $S \rightarrow B : M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}}$
4. $B \rightarrow A : M, \{N_a, K_{ab}\}_{K_{as}}$

where A and B are the two principals, K_{as} and K_{bs} are their private keys and S is the authentication server. The principals A and B generate the nonces N_a, N_b and M, and the server S generates K_{ab} which becomes the session key between A and B.

This description is expressed in the syntax of the *Protocol Analyzer*:

```
1) a -> b : m, a, b, enc(shared(kas)){[na, m, a, b]}
2) b -> s : m, a, b, enc(shared(kas)){[na, m, a, b]},
enc(shared(kbs)){[nb, m, a , b]}
3) s -> b : m, enc(shared(kas)){[na, shared(kab)]},
```

```
enc(shared(kbs)){[nb, shared (kab)]}
4) b -> a : m, enc(shared(kas)){[na, shared(kab)]}
```

As mentioned in the previous section, the above statements are written with the rules given in table 1 in section 5.2.1.

The initial assumptions of principals a, b and s are:

```
fact(5, possesses(a, shared(kas)), reason([],'Assumption')).
fact(6, believes(a, secret(a, kas, s)), reason([],'Assumption')).
fact(7, possesses(a, na), reason([],'Assumption')).
fact(8, believes(a, recognizes(na)), reason([], 'Assumption')).
fact(9, believes(a, fresh(na)), reason([], 'Assumption')).
fact(10, believes(a, honest(s)), reason([], 'Assumption')).
fact(11, believes(a, controls(s, secret(a, K, b))), reason([],
     'Assumption')).
fact(12, possesses(a, m), reason([], 'Assumption')).
fact(13, believes(a, fresh(m)), reason([], 'Assumption')).

fact(14, possesses(b, shared(kbs)), reason([],'Assumption')).
fact(15, believes(b, secret(b, kbs, s)), reason([],'Assumption')).
fact(16, possesses(b, nb), reason([],'Assumption')).
fact(17, believes(b, recognizes(nb)), reason([], 'Assumption')).
fact(18, believes(b, fresh(nb)), reason([], 'Assumption')).
fact(19, believes(b, honest(s)), reason([], 'Assumption')).
fact(20, believes(b, controls(s, secret(a, K, b))), reason([],
     'Assumption')).

fact(21, possesses(s, shared(kas)), reason([],'Assumption')).
```

```
fact(22, believes(s, secret(a, kas, s)), reason([],'Assumption')).
fact(23, possesses(s, shared(kbs)), reason([],'Assumption')).
fact(24, believes(s, secret(b, kbs, s)), reason([],'Assumption')).
fact(25, possesses(s, shared(kab)), reason([],'Assumption')).
fact(26, believes(s, secret(a, kab, b)), reason([],'Assumption')).

flag(count, 26).
```

These assumptions are written according to the syntax given in section 4.1.2.

When the *Protocol Analyzer* is provided with the above description and initial assumptions, it displays the schematic diagram for Otway-Rees protocol as shown in appendix B, section 3

When the full-run mode is invoked, the sequence of messages transmitted during the protocol are displayed in the schematic diagram.

Beliefs subwindows are displayed for each principal along with the main window displaying the schematic diagram. Figure 14 in appendix B shows a beliefs subwindow for principal s, listing the beliefs of s after step 2 of the protocol.

Modification of initial assumptions

In the case of Otway-Rees protocol, we show the effect of change of initial assumptions on the final beliefs of a, b and s.

Stepping through the Otway-Rees protocol to completion shows that since a believes in the freshness and possession of m, other conclusions m like believing in the freshness of message components containing m, follow.

Now the initial assumptions are changed and the initial beliefs of a, in freshness of m are deleted. It is observed that a does not believe in the freshness of any message components containing m, anymore.

The goal of the protocol is that a believes in the freshness of m and freshness of message components containing m. We can see from the above experiment that removal of initial beliefs of a in the freshness of m results in the goal not being attained.

The beliefs of a before and after the modification of the initial assumptions are shown in figures in appendix B, section 3. The above experiment also aids in understanding the role of particular initial assumptions in the final outcome of the protocol and the effect of their absence from the initial assumptions provided to the protocol analyzer.

With experiments performed on Voting and Otway-Rees protocols, we can see the usefulness of the *Protocol Analyzer* in the tedious process of protocol analysis and design.

Chapter 7

Conclusion

The main task in the project was to develop a tool serving two main objectives: a computer aided learning tool for showing executions of protocols and a front-end user interface for the automated GNY logic tool. This has been done and the system is tested for some known protocols.

In this chapter, possible extensions and enhancements to this system are discussed.

7.1 Extensions

During and after the process of development of the *Protocol Analyzer*, several useful extensions were thought of and these are described in this section. It is believed that these extensions could increase the usefulness of the system and also its applicability.

7.1.1 Full Protocol Simulation

The development of an automated learning tool for protocols followed in the *Protocol Analyzer* is modeled on the Interrogator [7]. The *Protocol Analyzer*

provides a display of the chosen protocol in the *Full-Run* mode. This is expected to help in the understanding of the working of the protocol. The approach followed has also concentrated on providing an interface to the automated GNY tool for a full formal analysis of the protocol. This is similar to the scheme used in the *Interrogator*, where a penetration attack by an intruder to obtain the value of a message data item, is found and displayed.

An alternative approach could have been followed in the development of the *Protocol Analyzer*. The emphasis could have been on an informal analysis of the protocol at the top level using the display features available and the schematic diagram. The following features would prove valuable in such an approach:

1. Provision of control and display of protocol messages as *bit strings*. This would enable the user to understand the role of every portion of the message in the bit string. This would prove useful as in an environment controlled by an intruder or if the channels between the principles were insecure, the attack would probably take place using the characteristics of bit strings of messages.

2. As an extension of the above concept, run-time values could be provided to protocol message components. This could include supplying user specified values to *variable* symbols, timestamps and nonces. A timestamp generation facility could also be included.

3. As far as encrypted or hashed function components of messages are concerned, a choice could be provided to the user at the message level to choose encryption or hash functions or even secret or public keys. A good method would be to display a menu of encryption or hash functions already implemented, to be used for a particular component in a

message.

The effect on the final outcome of the protocol as a result of choices made on the user's part could be observed in relation to any new vulnerability to attacks, introduced as a result. The above features could also aid in further understanding of the protocol working.

These features could be implemented within the current system itself, using the set of graphical user interface classes. These classes could be suitably modified to provide additional functionality and if required additional classes could be written for specific purposes.

7.2 Other Extensions

In relation to the current implementation of the *Protocol Analyzer*, several features could be incorporated considering existing shortcomings in it. These features are described below:

1. Although the text input and translation phase can handle 5 principals in the protocol, the display portion can display at most 3 principals. This limitation is because of implementation constraints and the potential complications that would result in the display portion. It would be a good extension to be able to provide for displaying protocols with more than 3 principals. A suitable display strategy for such protocols would also need to be worked out. Most of the changes would be in the *Control* module as it is the module which builds the display with fixed components and maintains it.

2. The syntax specified for protocol specification in the *Protocol Analyzer* cannot handle protocols with nested encryption or hashed components in

messages. A suitable extension would be to provide the nesting feature enabling a larger number of protocols to be analyzed by the analyzer. The level of nesting would obviously have to be limited to a number and this number explicitly specified. This feature could be provided by making changes in the class `TextInput` to both data members and member functions as needed.

3. The beliefs obtained and proofs of such derived beliefs, obtained by application of GNY logic are currently displayed as they are, in the same Prolog syntax. It would be a good enhancement to display beliefs and proofs in PostScript. A feature available with the automated GNY tool is to translate Prolog syntactic statements to Latex [26] format. The statements in Latex can then be converted to PostScript format using tools available in Unix like `latex` and `dvips`. This PostScript information can then be displayed using `ghostview`. The statements in PostScript appear in the standard syntax used for logics like BAN and GNY [4, 5]. This would facilitate better understanding of the positions of various principals.

4. It would be an ideal extension to provide a choice of protocols to analyze from within the *Protocol Analyzer*. A window pane listing protocols with their corresponding specification and initial assumptions' files, could be provided for the user to choose from. This would enable run-time changing of the protocol without exiting the system and rerunning it. This feature could be implemented using the existing classes.

5. The re-editing process for initial assumptions can be done only once. One enhancement could be to increase the scope of re-editing to provide more than one change, but obviously limited to some number. Another

related feature could be to provide a facility to go back to the original set of assumptions, discarding changes to the initial assumptions. An option could also be included to allow the user to specify a particular version of the assumptions, to go back to, in the case when more than one change in the assumptions is allowed.

The *Protocol Analyzer* in its current implementation nevertheless provides a good environment to observe the working of protocols and perform belief based analysis on them. It is expected to serve the needs of protocol designers in designing secure protocols for use in many authentication and key exchange applications.

References

[1] R.M.Needham and M.D.Schroeder, Using Encryption for Authentication in Large Networks of Computers, *CACM* Vol 21, No 12, pp 993-999, December 1978.

[2] D.Otway and O.Rees, Efficient and Timely Mutual Authentication, *Operating Systems Review*, Volume 21, January 1987.

[3] D.E.Denning and G.M.Sacco, Timestamps in Key Distribution Protocols, *CACM*, Number 8, pp533-536, volume 24, August 1981.

[4] M. Burrows and M. Abadi and R.M.Needham, *A Logic of Authentication*, Research Report 39, Digital Systems Research Center, Palo Alto, California, A condensed version of this report appeared in ACM Trans. on Computer Systems, 8(1):18-36, February 1990., February 1990.

[5] L.Gong and R.Needham and R.Yahalom, Reasoning about Belief in Cryptographic Protocols, *Proceedings of IEEE Symposium on Security and Privacy (Los Alamitos, California)*, IEEE Computer Society Press, pp234-248, 1990.

[6] D.Dolev and A.Yao, On the Security of Public Key Protocols, *IEEE Transactions on Information Theory*, 29(2), pp198-208, March 1983.

[7] J.K.Millen, S.C.Clark, and S.B.Freedman, The Interrogator : Protocol Security Analysis, *IEEE Transactions on Software Engineering*, SE-13(2), 1987.

[8] J.H.Moore, Protocol Failures in Cryptosystems, *Proceedings of the IEEE*, Vol.76, No. 5, May 1988.

[9] C.P.Pfleeger, *Security in Computing*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1989.

[10] CCITT, *CCITT Draft Recommendation X.509. The directory authentication framework*, Version 7, November 1987.

[11] Catherine A. Meadows, Formal Verification of Cryptographic Protocols: A Survey, *Pre-Proceedings of ASIACRYPT '94*, November 1994.

[12] Anish Mathuria, *Automating BAN Logic*, Master of Science(Honors) thesis, 1994.

[13] A. Mathuria, R. Safavi-Naini, P. Nickolas, *On the Automation of GNY Logic*, Technical Report, Department of Computer Science, University of Wollongong, 1994.

[14] Adrian Nye, *Xlib Programming Manual for Version 11*, O'Reilly & Associates Inc., 1993.

[15] Adrian Nye, *Xlib Reference Manual for Version 11*, O'Reilly & Associates Inc., 1993.

[16] G. J. Simmons, Cryptanalysis and Protocol Failures, *Communications of the ACM*, November 1994.

[17] M. Tatebayashi, N. Matsuzaki, D. B. Newman, Key distribution protocol for digital mobile communication systems. *Lecture Notes in Computer Science 435: Advances in Cryptology: Proceedings of Crypto '89*, Springer-Verlag, pp324-333, Berlin 1990.

[18] R.L. Rivest, A. Shamir, L.Adleman, A method for obtaining digital signatures and public key cryptosystems, *Communications of the ACM 21*, pp120-126, 1978.

[19] J. Hastad, On Using RSA with low exponent in a public key network, *Advances in Cryptology - Proceedings of Crypto '85*, pp403-408, 1985.

[20] G.J. Simmons, D.B. Holdridge, Forward search as a cryptanalytic tool against a public key privacy channel, *Proceedings of 1982 symposium on Security and Privacy*, pp117-128, 1982.

[21] P.S.Henry, Fast Implementation of the knapsack cipher, *B.S.T.J* vol 60, pp767-773, 1981.

[22] P.S. Henry and R.D. Nash, High speed hardware implementation of the knapsack cipher, *Presented at Crypto '81*, August 1981.

[23] National Bureau of Standards, Data Encryption Standard, U.S. Department of Commerce, *FIPS publication 46*, January 1977.

[24] Proposed Federal Standard 1026, Telecommunications: Interoperability and security replacements for the use of the data encryption standard in the physical and data link layers of data communications, *National Communications System, Washington DC draft*, June 1, 1981.

[25] W. Diffie, M. Hellman, New Directions in Cryptography, *IEEE Transactions in Information Theory*, 1976.

[26] L. Lamport, LaTeX: *A document preparation system.* Addison-Wesley publishing company, 1986.

[27] R.J. Anderson, Why cryptosystems fail, *Communications of the ACM*, November 1994.

[28] *The XSB Programmer's Manual Version 1.3*, Department of Computer Science, State University of New York at Stony Brook, September 1993.

[29] E. Campbell, R. Safavi-Naini, On Automating the BAN Logic of Authentication, *Proc. of the Fifteenth Australian Computer Science Conference, 1992.*

[30] S.B. Lippman, *C++ Primer*, Addison-Wesley Publishing Company, 1991.

[31] A.M. Mathuria, R. Safavi-Naini, P. Nickolas, On the Automation of GNY Logic, *Proc. of the Eighteenth Australasian Computer Science Conference, 1995.*

Appendix A

The Protocol Analyzer Source Code

```
//////////////////////////////////////////////////////////
//
// File : Arc.h
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the global declarations
// and class specification for class Arc.
//
//
//////////////////////////////////////////////////////////

#ifndef _DISPLAYBASE_
#include "DisplayBase.h"
#endif

#define EVENT_OFFSET 10

struct eqn
{
    float slope;
    float c;
};

class Arc : public DisplayBase
{
    public:
    Arc(Display *display, Window parent_win, GC gc,
        int pos_x_1, int pos_y_1, int pos_x_2, int pos_y_2);

    void DrawArc() const;
    void HighlightArc(arrowDir thearrowDir) const;
    void UnHighlightArc(arrowDir thearrowDir) const;
    Bool Pointer_in_area() const;

    private:
    void DrawArrow(arrowDir thearrowDir) const;
```

```
    int fpos_x_1;
    int fpos_y_1;
    int fpos_x_2;
    int fpos_y_2;
    struct eqn feqn;

};


////////////////////////////////////////////////////////
//
// File : Arc.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains member and other
// functions related to Class Arc.
//
//
////////////////////////////////////////////////////////

#include "Arc.h"

Arc::Arc(Display *display, Window parent_win, GC gc,
            int pos_x_1, int pos_y_1, int pos_x_2, int pos_y_2)
{
    fdisplay = display;
    fcurrent_window = parent_win;
    fgc = gc;
    fpos_x_1 = pos_x_1;
    fpos_y_1 = pos_y_1;
    fpos_x_2 = pos_x_2;
    fpos_y_2 = pos_y_2;

    // Determine and store the slope of the arc
    feqn.slope = (fpos_y_2 - fpos_y_1)/(fpos_x_2 - fpos_x_1);
    feqn.c = fpos_y_1 - (feqn.slope * fpos_x_1);
}

void
Arc::DrawArc() const
{
    // Go ahead and draw the arc(line)
    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_1, fpos_y_1,
                fpos_x_2, fpos_y_2);
    XFlush(fdisplay);
}

void
Arc::HighlightArc(arrowDir thearrowDir = right) const
{
    unsigned int line_width;

    // Change line width to 5 for drawing
```

```
    line_width = 5;
    XSetLineAttributes(fdisplay, fgc, line_width, LineSolid,
                       CapButt, JoinMiter);

    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_1, fpos_y_1,
              fpos_x_2, fpos_y_2);
    DrawArrow(thearrowDir);

    // Reset the line width
    line_width = 1;
    XSetLineAttributes(fdisplay, fgc, line_width, LineSolid,
                       CapButt, JoinMiter);
    XFlush(fdisplay);
}

void
Arc::DrawArrow(arrowDir thearrowDir) const
{
    switch (thearrowDir)
    {
    case right:
                if (fpos_y_1 < fpos_y_2)
                {
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_2,
                              fpos_y_2, fpos_x_2, fpos_y_2 - 30);
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_2,
                              fpos_y_2, fpos_x_2 + 40, fpos_y_2 - 15);
                }
                else
                if (fpos_y_1 > fpos_y_2)
                {
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_2,
                              fpos_y_2, fpos_x_2, fpos_y_2 + 30);
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_2,
                              fpos_y_2, fpos_x_2 + 40, fpos_y_2 + 15);
                }
                else
                {
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_2,
                              fpos_y_2, fpos_x_2 - 20, fpos_y_2 - 15);
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_2,
                              fpos_y_2, fpos_x_2 - 20, fpos_y_2 + 15);
                }
                break;

    case left:
                if (fpos_y_1 < fpos_y_2)
                {
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_1,
                              fpos_y_1, fpos_x_1, fpos_y_1 + 30);
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_1,
                              fpos_y_1, fpos_x_1 - 40, fpos_y_1 + 15);
                }
                else
                if (fpos_y_1 > fpos_y_2)
                {
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_1,
```

```
                                fpos_y_1, fpos_x_1 - 40, fpos_y_1 - 15);
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_1,
                                fpos_y_1, fpos_x_1, fpos_y_1 - 30);
                }
                else
                {
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_1,
                                fpos_y_1, fpos_x_1 + 20, fpos_y_1 - 15);
                    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_1,
                                fpos_y_1, fpos_x_1 + 20, fpos_y_1 + 15);
                }
                break;
    }
    XFlush(fdisplay);
}

void
Arc::UnHighlightArc(arrowDir thearrowDir = right) const
{
    unsigned int line_width;

    line_width = 5;
    XSetLineAttributes(fdisplay, fgc, line_width, LineSolid,
                        CapButt, JoinMiter);
    XSetFunction(fdisplay, fgc, GXclear);

    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_1, fpos_y_1,
                fpos_x_2, fpos_y_2);

    // Unhighlight the arrow
    DrawArrow(thearrowDir);

    XSetFunction(fdisplay, fgc, GXcopy);

    line_width = 1;
    XSetLineAttributes(fdisplay, fgc, line_width, LineSolid,
                        CapButt, JoinMiter);
    XDrawLine(fdisplay, fcurrent_window, fgc, fpos_x_1, fpos_y_1,
                fpos_x_2, fpos_y_2);
    XFlush(fdisplay);
}

Bool
Arc::Pointer_in_area() const
{
    Window root_win, child_win;
    int root_x, root_y, win_x, win_y;
    unsigned int mask;

    // Determine x and y position of mouse pointer
    if (XQueryPointer(fdisplay, fcurrent_window, &root_win, &child_win,
        &root_x, &root_y, &win_x, &win_y, &mask))
            if ((win_y ≥ (feqn.slope * win_x + feqn.c)) &&
                (win_y ≤ (feqn.slope * win_x + feqn.c + EVENT_OFFSET)))
                return True;

    return False;
```

```
}

//////////////////////////////////////////////////////
//
// File : BeliefsWindow.h
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the global declarations
// and class specification for class
// BeliefsWindow.
//
//////////////////////////////////////////////////////

#ifndef _DISPLAYBASE_
#include "DisplayBase.h"
#endif

#ifndef _SCROLLBAR_
#include "ScrollBar.h"
#endif

#include <ctype.h>

#define DELTA_X 10
#define NUM_BELIEF_LINES 200
#define NUM_PROOF_LINES 100

// Typedef for structure storing beliefs
typedef struct
{
    char *beliefs[NUM_BELIEF_LINES];
    unsigned short wraps[NUM_BELIEF_LINES];
    int num_beliefs;
} beliefs_text;


class BeliefsWindow : public DisplayBase
{
    public:
    BeliefsWindow(Display *display, Window parent_win, int margc,
                  char *margv[], GC gc,
                  char *windowname, int left_x, int left_y,
                  unsigned int width, unsigned int height,
                  unsigned int border_width, unsigned long border,
                  unsigned long background, char *fontname);
    ~BeliefsWindow();

    Window Window_id() { return fcurrent_window; };
    void Init_y_offset() { fy_dist = INCREMENT_Y; };
    void DrawItself();
    void LoadBeliefs(char *text);
    char *Handle_click(unsigned int button);
    void DisplayBeliefs(Direction show);
    void DestroyBeliefs();
```

```
    private:
    char *Remove_extensions(char *btext);

    Window fparent_window;
    int fleft_x;
    int fleft_y;
    unsigned int fwidth;
    unsigned int fheight;
    unsigned int fborder_width;
    unsigned long fborder;
    unsigned long fbackground;
    unsigned int fx_dist;
    unsigned int fy_dist;
    XFontStruct *ffont_ptr;
    beliefs_text theBeliefstext1;
    unsigned short top_display_index;
    unsigned short bottom_display_index;
    ScrollBar *sbar;
};

////////////////////////////////////////////////////////////
//
// File : BeliefsWindow.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains member and other
// functions for class BeliefsWindow.
//
//
////////////////////////////////////////////////////////////

#include "BeliefsWindow.h"

void
inline strpartcpy(char *string1, char *string2, short start,
                  short end)
{
    short i, j;

    for (i=start,j=0;i<(start + end);i++,j++)
        string1[j] = string2[i];
}

inline char *
strip_facts(char *string)
{
    char *dum_str;
    char *end_ptr;
    char *temp_str;

    if (!strstr(string, FACT))
        return string;
```

```
    temp_str = new char[strlen(string) + 1];
    strcpy(temp_str, string);

    dum_str = strchr(temp_str, ',');

    if (dum_str ≠ NULL)
    {
        dum_str++;
        while (isspace(*dum_str))
            dum_str++;
    }

    temp_str = dum_str;

    end_ptr = strrchr(temp_str,')');

    *end_ptr = NULL;

    return temp_str;
}

BeliefsWindow::BeliefsWindow(Display *display, Window parent_win,
                             int margc, char *margv[],
                             GC gc, char *windowname, int left_x,
                             int left_y, unsigned int width,
                             unsigned int height, unsigned int border_width,
                             unsigned long border, unsigned long background,
                             char *fontname)
{
    XSizeHints xsh;
    XWMHints wm_hints;
    XClassHint class_hints;
    XTextProperty windowName, iconName;
    char *iconname;

    fdisplay = display;
    fparent_window = parent_win;
    fgc = gc;
    iconname = windowname;
    fleft_x = left_x;
    fleft_y = left_y;
    fwidth = width;
    fheight = height;
    fborder_width = border_width;
    fborder = border;
    fbackground = background;
    fx_dist = DELTA_X;

    // Create the window and store its window id
    fcurrent_window = XCreateSimpleWindow(fdisplay, fparent_window,
                                          fleft_x, fleft_y, fwidth,
                                          fheight, fborder_width,
                                          fborder, fbackground);

    xsh.flags = (PPosition | PSize);
    xsh.height = fheight;
    xsh.width = fwidth;
```

```
    xsh.x = fleft_x;
    xsh.y = fleft_y;

    wm_hints.initial_state = NormalState;
    wm_hints.flags = StateHint;

    class_hints.res_name = windowname;
    class_hints.res_class = "Secure";

    XStringListToTextProperty(&windowname, 1, &windowName);
    XStringListToTextProperty(&windowname, 1, &iconName);

    // Select expose and buttonpress events for the window
    XSelectInput(fdisplay, fcurrent_window, ExposureMask | ButtonPressMask);

    // Create scrollbar instance
    sbar = new ScrollBar(fdisplay, fcurrent_window, fgc, 1,
                                1, SCROLLBAR_WIDTH, fheight - 2,
                                SCROLLBAR_WIDTH + 2, INCREMENT_Y + 10 );

    XMapWindow(fdisplay, fcurrent_window);

    // Hints to window manager
    XSetWMProperties(fdisplay, fcurrent_window, &windowName,
                        &iconName, margv, margc, &xsh, &wm_hints,
                        &class_hints);

    if ((ffont_ptr = XLoadQueryFont(fdisplay, fontname)) == NULL)
    {
        cerr << "cannot load font :  " << fontname << "\n";
        cout.flush();
    }
    XSetFont(fdisplay, fgc, ffont_ptr->fid);

    // Initialize class variables
    theBeliefstext1.num_beliefs = -1;
    top_display_index = 0;
    bottom_display_index = 0;
}

BeliefsWindow::~BeliefsWindow()
{
    short i;

    // Delete the belief strings
    for(i=0; i < theBeliefstext1.num_beliefs; i++)
        delete theBeliefstext1.beliefs;

    delete ffont_ptr;

    // Delete the scrollbar
    delete sbar;
    XDestroyWindow(fdisplay, fcurrent_window);
}

void
BeliefsWindow::DrawItself()
```

```
{
    XClearWindow(fdisplay, fcurrent_window);

    // Draw the scroll bar in the beliefs window
    sbar→DrawButton();

    XFlush(fdisplay);
}

void
BeliefsWindow::LoadBeliefs(char *text)
{
    // Load the belief
    theBeliefstext1.num_beliefs += 1;
    theBeliefstext1.beliefs[theBeliefstext1.num_beliefs] =
                new char[strlen(text)+1];
    strcpy(theBeliefstext1.beliefs[theBeliefstext1.num_beliefs],
            text);

    // Initialize the wrap count for that line
    theBeliefstext1.wraps[theBeliefstext1.num_beliefs] = 0;
}

void
BeliefsWindow::DestroyBeliefs()
{
    short k;

    // Deallocate existing belief strings
    for (k=0;k≤theBeliefstext1.num_beliefs;k++)
        delete theBeliefstext1.beliefs[k];

    // Set number of beliefs back to -1
    theBeliefstext1.num_beliefs = -1;

}

char *
BeliefsWindow::Handle_click(unsigned int button)
{
    Window root_win, child_win;
    int root_x, root_y, win_x, win_y, temp_y, store;
    unsigned int mask;
    unsigned short k, j;

    // Check if mouse click in scrollbar
    if (sbar→Pointer_in_area())
    {

        switch(button)
        {
        case 1:
                if (bottom_display_index < theBeliefstext1.num_beliefs)
                {
                    //Let the scrollbar do the scrolling first
                    sbar→Handle_click(button);
```

```
                    //Display the belief after scrolling down
                    DisplayBeliefs(down);
                }
                break;

        case 2:
                if (top_display_index > 0)
                {
                    //Let the scrollbar do the scrolling first
                    sbar→Handle_click(button);

                    //Display the belief after scrolling up
                    DisplayBeliefs(up);
                }
                break;
        }
    }
    else
    {
        // Else where else on the window the mouse is ?
        XQueryPointer(fdisplay, fcurrent_window, &root_win, &child_win,
                &root_x, &root_y, &win_x, &win_y, &mask);

        temp_y = win_y/INCREMENT_Y + top_display_index;

        // Get the right line, considering wraps of line above the
        // one on which clicked
        store = temp_y;
        for (k = top_display_index, j = top_display_index; k < (store-1); j++)
        {
            temp_y -= theBeliefstext1.wraps[j];
            theBeliefstext1.wraps[j] ? k += theBeliefstext1.wraps[j] : k++;
        }

        // Finally, return belief clicked on
// if (strstr(theBeliefstext1.beliefs[temp_y], FACT))
            return theBeliefstext1.beliefs[temp_y];
    }
    return NULL;
}


void
BeliefsWindow::DisplayBeliefs(Direction show)
{
    XTextItem *text_item;
    unsigned short unit_text_width;
    unsigned short no_chars, i;
    short leftover, copy_from;
    short index;
    unsigned short num_chars;
    char *output_text;
    unsigned int temp_y_dist;

    // Text width per character
    unit_text_width = XTextWidth(ffont_ptr, strip_facts(theBeliefstext1.beliefs[0]), 1);
    num_chars = fwidth/unit_text_width;
```

```
switch(show)
{
case all:

                // Initial display of beliefs
                DrawItself();

                if (theBeliefstext1.num_beliefs > -1)
                {
                for (i=0; i≤theBeliefstext1.num_beliefs; i++)
                {
                    leftover = strlen(strip_facts(theBeliefstext1.beliefs[i]));
                    copy_from = 0;

                    // Till no characters left in belief...
                    while ((leftover > 0) && (fy_dist < fheight))
                    {
                        no_chars = leftover > num_chars ? num_chars : leftover;
                        text_item = new XTextItem[sizeof(XTextItem)];
                        text_item→chars = new char[no_chars + 1];
                        strpartcpy(text_item→chars,
                                    strip_facts(theBeliefstext1.beliefs[i]), copy_from,
                                    no_chars);
                        leftover -= num_chars;

                        // If going to wrap the line, increment its wrap count
                        if (leftover > 0)
                            theBeliefstext1.wraps[i] += 1;
                        copy_from += no_chars;
                        text_item→nchars = no_chars;
                        text_item→delta = DELTA_X;
                        text_item→font = None;
                        XDrawText(fdisplay, fcurrent_window, fgc,
                                    fx_dist, fy_dist, text_item, 1);
                        fy_dist += INCREMENT_Y;
                        delete text_item;
                    }

                    if (fy_dist ≥ fheight)
                    {
                        top_display_index = 0;
                        bottom_display_index = i;
                        return;
                    }

                }
                top_display_index = 0;
                bottom_display_index = i - 1;
                }
                break;

case up:
case down:
                // Redisplay beliefs during up and down scrolling
                if (show == up)
                {
```

```
                    top_display_index--;
                    bottom_display_index--;
                    index = top_display_index;
                }
                if (show == down)
                {
                    top_display_index++;
                    bottom_display_index++;
                    index = bottom_display_index;
                }

// fy_dist -= INCREMENT_Y;
                copy_from = 0;
                leftover = strlen(strip_facts(theBeliefstext1.beliefs[index]));
                while (leftover > 0)
                {
                    no_chars = leftover > num_chars ? num_chars : leftover;
                    text_item = new XTextItem[sizeof(XTextItem)];
                    text_item→chars = new char[no_chars + 1];
                    strpartcpy(text_item→chars,
strip_facts(theBeliefstext1.beliefs[index]),
                                copy_from, no_chars);
                    leftover -= num_chars;
                    copy_from += no_chars;
                    text_item→nchars = no_chars;
                    text_item→delta = DELTA_X;
                    text_item→font = None;
                    XDrawText(fdisplay, fcurrent_window, fgc, fx_dist, fy_dist,
                                text_item, 1);
                    XFlush(fdisplay);
                    fy_dist += INCREMENT_Y;
                    delete text_item;
                }
                break;

        case retain:
                    // For expose handling on the window
                    DrawItself();

                    for (i=top_display_index; i≤bottom_display_index; i++)
                    {
                        leftover = strlen(strip_facts(theBeliefstext1.beliefs[i]));
                        copy_from = 0;
                        while ((leftover > 0) && (fy_dist < fheight))
                        {
                            no_chars =
                            leftover > num_chars ? num_chars : leftover;
                            text_item = new XTextItem[sizeof(XTextItem)];
                            text_item→chars = new char[no_chars + 1];
                            strpartcpy(text_item→chars,
                                        strip_facts(theBeliefstext1.beliefs[i]),
                                        copy_from, no_chars);
                            leftover -= num_chars;
                            copy_from += no_chars;
                            text_item→nchars = no_chars;
                            text_item→delta = DELTA_X;
                            text_item→font = None;
```

```
                                XDrawText(fdisplay, fcurrent_window, fgc, fx_dist,
                                        fy_dist, text_item, 1);
                                fy_dist += INCREMENT_Y;
                                delete text_item;
                            }
                        }
                        break;
        }
}


////////////////////////////////////////////////////////
//
// File : Button.h
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the global declarations
// and class specification for class
// Button.
//
////////////////////////////////////////////////////////

#define _BUTTON_

#ifndef _DISPLAYBASE_
#include "DisplayBase.h"
#endif

#define TEXT_Y_OFFSET 28

class Button : public DisplayBase
{
    public:
    Button(Display *display, Window parent_win, GC gc,
            int left_x, int left_y, unsigned int width,
            unsigned int height, char *fontname, char *text);
    ~Button() { delete ftext; delete ffont_ptr; };
    void DrawButton() const;
    void DrawButtonText() const;
    Bool Pointer_in_area() const;
    virtual void Do_click();
    void ClearButtonArea() const;

    protected:
    int fleft_x;
    int fleft_y;
    unsigned int fwidth;
    unsigned int fheight;
    char *ftext;
    XFontStruct *ffont_ptr;
};

////////////////////////////////////////////////////////
//
```

```
// File : Button.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains member and other
// functions for class Button
//
//
////////////////////////////////////////////////////////////

#include "Button.h"

inline Bool
In_button_area(int lpos_x, int lpos_y, int wd, int ht, int pos_x,
               int pos_y)
{
    // Are the coordinates within the rectangular area
    if ((pos_x ≤ (lpos_x + wd)) && (pos_x ≥ lpos_x) &&
        (pos_y ≤ (lpos_y + ht)) && (pos_y ≥ lpos_y))
        return True;

    // No, return false
    return False;
}

Button::Button(Display *display, Window parent_win, GC gc,
               int left_x, int left_y, unsigned int width,
               unsigned int height, char *fontname, char *text)
{

    fdisplay = display;
    fcurrent_window = parent_win;
    fgc = gc;

    if (text ≠ NULL)
    {
        // Store button text
        ftext = new char[strlen(text)];
        strcpy(ftext, text);
    }

    fleft_x = left_x;
    fleft_y = left_y;
    fwidth = width;
    fheight = height;

    if (fontname ≠ NULL)
    {
        if ((ffont_ptr = XLoadQueryFont(fdisplay, fontname)) == NULL)
        {
            cerr ≪ "cannot load font :  " ≪ fontname ≪ "\n";
            cout.flush();
        }
        XSetFont(fdisplay, fgc, ffont_ptr→fid);
    }
```

```
}

void
Button::DrawButton() const
{

    // Actually draw the rectangle
    XDrawRectangle(fdisplay, fcurrent_window, fgc, fleft_x, fleft_y,
                        fwidth, fheight);

    XFlush(fdisplay);
}

void
Button::DrawButtonText() const
{
    int x_position;
    int text_width;

    // Calculate position for text within the button
    text_width = XTextWidth(ffont_ptr, ftext, strlen(ftext));
    x_position = fleft_x + fwidth/2 - text_width/2;

    // Draw the text within the button
    XDrawImageString(fdisplay, fcurrent_window, fgc, x_position,
                            fleft_y + TEXT_Y_OFFSET, ftext, strlen(ftext));
}

Bool
Button::Pointer_in_area() const
{
    Window root_win, child_win;
    int root_x, root_y, win_x, win_y;
    unsigned int mask;

    if (XQueryPointer(fdisplay, fcurrent_window, &root_win, &child_win,
        &root_x, &root_y, &win_x, &win_y, &mask))
            if (In_button_area(fleft_x, fleft_y, fwidth, fheight,
                                    win_x, win_y))
                return True;

    return False;
}

void
Button::Do_click()
{
    unsigned long foreground, background;
    int x_position;
    int text_width;

    text_width = XTextWidth(ffont_ptr, ftext, strlen(ftext));
    x_position = fleft_x + fwidth/2 - text_width/2;

    foreground = 1;
    background = 0;
    XSetForeground(fdisplay, fgc, foreground);
```

```
    XSetBackground(fdisplay, fgc, background);

    // Fill the button with background colour
    XFillRectangle(fdisplay, fcurrent_window, fgc,
                        fleft_x, fleft_y, fwidth, fheight);

    // Redraw the text
    XDrawImageString(fdisplay, fcurrent_window, fgc, x_position,
                        fleft_y + TEXT_Y_OFFSET, ftext, strlen(ftext));

    XFlush(fdisplay);
}

void
Button::ClearButtonArea() const
{
    Bool exposures;

    // Clear the button area, before a redraw
    exposures = True;
    XClearArea(fdisplay, fcurrent_window, fleft_x, fleft_y, fwidth,
                fheight, exposures);
}

//////////////////////////////////////////////////////////
//
// File : Control.h
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the global and
// forward declarations for the Control
// module.
//
//////////////////////////////////////////////////////////

#ifndef _DISPLAYBASE_
#include "DisplayBase.h"
#endif

#include "Arc.h"
#include "Node.h"
#include "Text.h"
#include "TextInput.h"

#ifndef _BUTTON_
#include "Button.h"
#endif

#ifndef _SCROLLBAR_
#include "ScrollBar.h"
#endif

#include "BeliefsWindow.h"
#include "MainWindow.h"
```

```
#include "StatusLine.h"
#include "Gny.h"

// Forward Declarations
void CreateDisplayAndLoop(int argc, char *argv[]);
void MessageXmitDisplay(MainWindow *progWin, Arc *theArc, char *text,
                        arrowDir direction);
void
MessageXmitHighlight(MainWindow *progWin, Arc *theArc, Text *SText,
                        TextInput *theTextInstance, arrowDir *dirntable,
                        unsigned short flag);
void DisplayProofs(BeliefsWindow *proofsWindow, char *string, Bool
                    Mode,int argc, char *argv[]);

void strpartcpy(char *string1, char *string2, short start,
                        short end);

// Control # defines
#define CREATE 1
#define DRAW 2
#define WINDOW_ID 3
#define HIGHLIGHT 1
#define UNHIGHLIGHT 2
#define BELIEFS_LIMIT 2
#define TEXT_INCREMENT_Y 20
#define SUPER_TEXT_LENGTH 70
#define MAX_TEXT_LENGTH 35
#define FONTNAME "*helvetica-bold-r*140*"

// Globals
Display *tdisplay;
GC gc;
short Go_loop_count;
char *common_font = FONTNAME;
Arc *arc_AS;
Arc *arc_SB;
Arc *arc_AB;
char assfile[40];

////////////////////////////////////////////////////////
//
// File : Control.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the main program
// of the Protocol Analyzer and functions
// in the Control module.
//
////////////////////////////////////////////////////////

#include "Control.h"

int
main(int argc, char *argv[])
```

```
{
    char *display_name = NULL;

    if (argc < 3)
    {
        cerr << "Invalid argument(s) \n";
        cerr << "Usage :  secure <messages filename> <assumptions filename>
\n";
        exit(1);
    }

    // Open connection to X server
    if ((tdisplay = XOpenDisplay(display_name)) == NULL)
        cerr << argv[0] << " :  Cannot connect to X server"
            << XDisplayName(display_name) << "\n";

    // A default GC for all graphics
    gc = XDefaultGC(tdisplay, XDefaultScreen(tdisplay));

    //Create all the components, display them and wait in an
    //event loop
    CreateDisplayAndLoop(argc, argv);

    // Close connection to X server
    XCloseDisplay(tdisplay);
    return 0;
}

void
CreateDisplayAndLoop(int argc, char *argv[])
{
    char *sl_text = "Welcome to the Secure Protocol Analyzer";
    char *exit_text = "Protocol run over.  Exiting !!";
    XEvent report;
    unsigned short i, no_beliefs, j;
    char *str, step_text[30];
    unsigned short num_nodes, beliefs_count;
    Node *node_S;
    Text *SampleText, *SText;
    char *t_str, *ret_str;
    unsigned short x_posn;
    BeliefsWindow *node_A_bw[BELIEFS_LIMIT];
    BeliefsWindow *node_B_bw[BELIEFS_LIMIT];
    BeliefsWindow *node_S_bw[BELIEFS_LIMIT];
    BeliefsWindow *proofsWindow;
    arrowDir *dirntable;
    char fileName[15];
    Gny *theGnyA, *theGnyB, *theGnyS;
    Bool displayed_all_a = False;
    Bool displayed_all_b = False;
    Bool displayed_all_s = False;
    Bool reedit = False, entered_reedit = False;
    Bool proofMode = False;
    char nodes[MAX_NODES][ENTITY_LENGTH];
    unsigned short node_cnt;
    MessageStruct *msgStruct;
    char *tstr;
```

```
char local_str[20], cpstr[20];
TextInput *textInstance;

// Name of assumptions file
strcpy(assfile, argv[2]);

// Create text instance object
textInstance = new TextInput(argv[1], assfile);
textInstance→LoadMessages();
textInstance→Transform_to_idealized();
textInstance→Dump_to_file();
num_nodes = textInstance→return_num_nodes();

// Load the entity names
node_cnt = 0;
tstr = textInstance→Return_sender_name();
while (tstr ≠ NULL)
{
    strcpy(nodes[node_cnt++], tstr);
    tstr = textInstance→Return_sender_name();
}

//Initialize beliefs set count
beliefs_count = 0;

// Create the main program window
MainWindow program_window(argc, argv,
                          DefaultRootWindow(tdisplay), gc,
                          argv[1], argv[1], tdisplay,
                          BASE_WD, BASE_HT, BASE_X, BASE_Y);

// Create the status line
StatusLine line(tdisplay, program_window.base_window(), gc, common_font,
                1, BASE_HT - 25, BASE_WD, 25);

// Create Button "GO"
Button Button_Go(tdisplay, program_window.base_window(),
                 gc, 100, 50, BUTTON_WD, BUTTON_HT, common_font,
                 "GO");

// Create Button "RUN"
Button Button_Run(tdisplay, program_window.base_window(),
                 gc, 250, 50, BUTTON_WD, BUTTON_HT, common_font,
                 "RUN");

// Create Button "RE-EDIT"
Button Button_Reedit(tdisplay, program_window.base_window(),
                 gc, 400, 50, BUTTON_WD, BUTTON_HT, common_font,
                 "RE-EDIT");

// Create Button "QUIT"
Button Button_Quit(tdisplay, program_window.base_window(),
                 gc, 550, 50, BUTTON_WD, BUTTON_HT, common_font,
                 "QUIT");

// Create Server node if required ...
if (num_nodes == 3)
```

```
    {
        node_S = new Node(tdisplay,
                program_window.base_window(), gc, common_font, nodes[2],
                350, 200,
                NODE_WIDTH, NODE_WIDTH, CIRCLE_START, CIRCLE_END);

        arc_AS = new Arc(tdisplay,
                program_window.base_window(), gc, 360, 255, 295, 335);

        arc_SB = new Arc(tdisplay,
                program_window.base_window(), gc, 470, 335, 400, 255);
    }

    // Create Node A
    Node node_A(tdisplay,
                program_window.base_window(), gc, common_font, nodes[0],
                250, 330, NODE_WIDTH, NODE_WIDTH, CIRCLE_START,
CIRCLE_END);

    // Create Node B
    Node node_B(tdisplay,
                program_window.base_window(), gc, common_font,
                nodes[1], 450, 330, NODE_WIDTH, NODE_WIDTH, CIRCLE_START,
                CIRCLE_END);

    // Create Arc AB
    arc_AB = new Arc(tdisplay,
                program_window.base_window(), gc, 250 + NODE_WIDTH, 360,
                450, 360);

    // Create Beliefs Window for Node A
    sprintf(local_str,"Beliefs of %s",nodes[0]);
    node_A_bw[beliefs_count] = new BeliefsWindow(tdisplay,
                    DefaultRootWindow(tdisplay),
                    argc, argv,
                    gc, local_str, 100, 300, 400 , 300, 1,
                    BlackPixel(tdisplay,
                    DefaultScreen(tdisplay)),
                    WhitePixel(tdisplay,
                    DefaultScreen(tdisplay)), common_font);

    // Create Beliefs Window for Node B
    sprintf(local_str,"Beliefs of %s",nodes[1]);
    node_B_bw[beliefs_count] = new BeliefsWindow(tdisplay,
                    DefaultRootWindow(tdisplay),
                    argc, argv,
                    gc, local_str, 700, 500, 400 , 300, 1,
                    BlackPixel(tdisplay,
                    DefaultScreen(tdisplay)),
                    WhitePixel(tdisplay,
                    DefaultScreen(tdisplay)), common_font);

    // Create Beliefs Window for Node S if need be...
    if (num_nodes == 3)
    {
                sprintf(local_str,"Beliefs of %s",nodes[2]);
                node_S_bw[beliefs_count] = new BeliefsWindow(tdisplay,
```

```
                    DefaultRootWindow(tdisplay),
                    argc, argv,
                    gc, local_str, 200, 600, 400 , 300, 1,
                    BlackPixel(tdisplay,
                    DefaultScreen(tdisplay)),
                    WhitePixel(tdisplay,
                    DefaultScreen(tdisplay)), common_font);
}

// Get the direction of arrows from text input object
dirntable = textInstance→ReturnArrowsTable();

// Initialize y direction offset for beliefs' display
node_A_bw[beliefs_count]→Init_y_offset();
node_B_bw[beliefs_count]→Init_y_offset();
if (num_nodes == 3)
    node_S_bw[beliefs_count]→Init_y_offset();

// Display the belief subwindows
node_A_bw[beliefs_count]→DrawItself();
node_B_bw[beliefs_count]→DrawItself();
if (num_nodes == 3)
    node_S_bw[beliefs_count]→DrawItself();

// Create GNY object for A
theGnyA = new Gny(nodes[0], assfile);

// Create GNY object for B
theGnyB = new Gny(nodes[1], assfile);

// Create GNy object for S
if (num_nodes == 3)
    theGnyS = new Gny(nodes[2], assfile);

// Parse output file for GNY objects
theGnyA→Gny_parse_file(assfile, BELIEFS);
theGnyB→Gny_parse_file(assfile, BELIEFS);
if (num_nodes == 3)
    theGnyS→Gny_parse_file(assfile, BELIEFS);

// Load beliefs into belief window objects
no_beliefs = theGnyA→Gny_num_beliefs();
for (i=0;i≤(no_beliefs-1);i++)
{
    str = theGnyA→Gny_return_current_belief();
    node_A_bw[beliefs_count]→LoadBeliefs(str);
}

no_beliefs = theGnyB→Gny_num_beliefs();
for (i=0;i≤(no_beliefs-1);i++)
{
    str = theGnyB→Gny_return_current_belief();
    node_B_bw[beliefs_count]→LoadBeliefs(str);
}

if (num_nodes == 3)
{
```

```
        no_beliefs = theGnyS→Gny_num_beliefs();
        for (i=0;i≤(no_beliefs-1);i++)
        {
            str = theGnyS→Gny_return_current_belief();
            node_S_bw[beliefs_count]→LoadBeliefs(str);
        }
}

// Delete existing GNY objects
delete theGnyA;
delete theGnyB;
if (num_nodes == 3)
    delete theGnyS;

//Single step count initialization
Go_loop_count = -1;

// Program's event loop...
// Never say die.....
while(TRUE)
{
    // Get next X event from main window or subwindows
    XNextEvent(tdisplay, &report);

    switch (report.type)
    {
    case ButtonPress:
                    // Clicked on QUIT ?
                    if (Button_Quit.Pointer_in_area())
                    {
                        // Okay, its time to say good bye...
                        line.ClearTextArea();
                        line.DisplayText(exit_text);
                        return;
                    }

                    // Clicked on GO ?
                    if (Button_Go.Pointer_in_area())
                    {
                        // Great, its single-stepping now

                        Button_Go.Do_click();

                        // One step over
                        Go_loop_count =
                            (Go_loop_count+1) % textInstance→NumMessages();

                        // If after re-editing initial assumptions
                        if (entered_reedit)
                        {

                            beliefs_count++;

                            // Create Beliefs Window for Node A
                            sprintf(local_str,
                                    "Revised beliefs of %s",nodes[0]);
                            node_A_bw[beliefs_count] =
```

```
            new BeliefsWindow(tdisplay,
                DefaultRootWindow(tdisplay),
                argc, argv,
                gc, local_str, 50, 500, 400,
                300, 1, BlackPixel(tdisplay,
                DefaultScreen(tdisplay)),
                WhitePixel(tdisplay,
                DefaultScreen(tdisplay)),
                common_font);

// Create Beliefs Window for Node B
sprintf(local_str,
    "Revised beliefs of %s",nodes[1]);
node_B_bw[beliefs_count] =
    new BeliefsWindow(tdisplay,
        DefaultRootWindow(tdisplay),
        argc, argv,
        gc, local_str, 600, 600, 400,
        300, 1, BlackPixel(tdisplay,
        DefaultScreen(tdisplay)),
        WhitePixel(tdisplay,
        DefaultScreen(tdisplay)), common_font);

// Create Beliefs Window for Node S if need be..
if (num_nodes == 3)
{
        sprintf(local_str,
            "Revised beliefs of %s",nodes[2]);
        node_S_bw[beliefs_count] =
            new BeliefsWindow(tdisplay,
            DefaultRootWindow(tdisplay),
            argc, argv,
            gc, local_str, 100, 300,
            400 , 300, 1,
            BlackPixel(tdisplay,
            DefaultScreen(tdisplay)),
            WhitePixel(tdisplay,
            DefaultScreen(tdisplay)),
            common_font);
}

node_A_bw[beliefs_count]→Init_y_offset();
node_B_bw[beliefs_count]→Init_y_offset();
if (num_nodes == 3)
    node_S_bw[beliefs_count]→Init_y_offset();

node_A_bw[beliefs_count]→DrawItself();
node_B_bw[beliefs_count]→DrawItself();
if (num_nodes == 3)
    node_S_bw[beliefs_count]→DrawItself();

Go_loop_count = -1;

// Set entered flag to false as we no
// longer need initial actions done above
entered_reedit = False;
```

```
    delete textInstance;
    textInstance = new TextInput(argv[1], assfile);
    textInstance→LoadMessages();
    textInstance→Transform_to_idealized();
    textInstance→Dump_to_file();
    num_nodes = textInstance→return_num_nodes();
                                    // Initialize message counter to 0
// textInstance→Reinitialize_message_count();

                                }

                                // Is it still okay to single-step
                                if (Go_loop_count < textInstance→NumMessages())
                                {
                                    sprintf(step_text,
                                    "Single Step option :  Step %d",
                                        Go_loop_count+1);
                                    line.ClearTextArea();
                                    line.DisplayText(step_text);

                                    // Get next message
                                    msgStruct = textInstance→NextMessage();
                                    node_A_bw[beliefs_count]→Init_y_offset();
                                    node_B_bw[beliefs_count]→Init_y_offset();
                                    if (num_nodes == 3)
                                        node_S_bw[beliefs_count]→Init_y_offset();
                                    SText = new Text(tdisplay,
                                                program_window.base_window(), gc,
                                                common_font, 320, 275,
                                                msgStruct→message);

                                    strcpy(fileName,
                                        textInstance→Return_msg_file());

                                    // Create GNY object for A
                                    theGnyA = new Gny(nodes[0], assfile);

                                    // Create GNY object for B
                                    theGnyB = new Gny(nodes[1], assfile);

                                    if (num_nodes == 3)
                                    // Create GNY object for B
                                        theGnyS = new Gny(nodes[2], assfile);

                                    // Destroy existing beliefs in A and B
                                    node_A_bw[beliefs_count]→DestroyBeliefs();

                                    // Load Beliefs into node A's window object
                                    theGnyA→Gny_get_beliefs_and_proofs(fileName,
                                            BELIEFS, NULL);
                                    no_beliefs = theGnyA→Gny_num_beliefs();
                                    for (i=0;i≤(no_beliefs-1);i++)
                                    {
                                        str = theGnyA→Gny_return_current_belief();
                                        node_A_bw[beliefs_count]→LoadBeliefs(str);
                                    }
```

```
node_A_bw[beliefs_count]→DrawItself();
node_A_bw[beliefs_count]→DisplayBeliefs(all);

// Destroy B's beliefs
node_B_bw[beliefs_count]→DestroyBeliefs();

// Load Beliefs into node B's window object
theGnyB→Gny_get_beliefs_and_proofs(fileName,
            BELIEFS, NULL);
no_beliefs = theGnyB→Gny_num_beliefs();
for (i=0;i≤(no_beliefs-1);i++)
{
    str = theGnyB→Gny_return_current_belief();
    node_B_bw[beliefs_count]→LoadBeliefs(str);
}

node_B_bw[beliefs_count]→DrawItself();
node_B_bw[beliefs_count]→DisplayBeliefs(all);

if (num_nodes == 3)
{
    // Destroy S's beliefs
    node_S_bw[beliefs_count]→DestroyBeliefs();

    // Load Beliefs into node B's window object
    theGnyS→Gny_get_beliefs_and_proofs(
            fileName, BELIEFS, NULL);
    no_beliefs = theGnyS→Gny_num_beliefs();
    for (i=0;i≤(no_beliefs-1);i++)
    {
        str =
        theGnyS→Gny_return_current_belief();
        node_S_bw[beliefs_count]→LoadBeliefs(
            str);
    }

    node_S_bw[beliefs_count]→DrawItself();
    node_S_bw[beliefs_count]→DisplayBeliefs(
        all);

}

// Display arc AB → direction
if ((!strcmp(msgStruct→sender, nodes[0]))
    && (!strcmp(msgStruct→receiver, nodes[1])))
MessageXmitDisplay(&program_window,
    arc_AB, msgStruct→message, right);

// Display arc AB <- direction
if ((!strcmp(msgStruct→receiver, nodes[0]))
    && (!strcmp(msgStruct→sender, nodes[1])))
MessageXmitDisplay(&program_window,
    arc_AB, msgStruct→message, left);
if (num_nodes == 3)
{
    // arc SB →
    if ((!strcmp(msgStruct→sender, nodes[1]))
```

```
                    && (!strcmp(msgStruct→receiver,
                    nodes[2])))
                    MessageXmitDisplay(&program_window,
                        arc_SB, msgStruct→message, right);

                // arc SB <-
                if ((!strcmp(msgStruct→receiver, nodes[1]))
                    && (!strcmp(msgStruct→sender,
                    nodes[2])))
                    MessageXmitDisplay(&program_window,
                        arc_SB, msgStruct→message, left);

                // arc AS →
                if ((!strcmp(msgStruct→sender, nodes[2]))
                    && (!strcmp(msgStruct→receiver,
                    nodes[0])))
                    MessageXmitDisplay(&program_window,
                        arc_AS, msgStruct→message, right);

                // arc AS <-
                if ((!strcmp(msgStruct→receiver, nodes[2]))
                        && (!strcmp(msgStruct→sender,
                        nodes[0])))
                    MessageXmitDisplay(&program_window,
                        arc_AS, msgStruct→message, left);
            }
            XFlush(tdisplay);

            // Delete current GNY objects
            delete theGnyA;
            delete theGnyB;
            if (num_nodes == 3)
                delete theGnyS;
        }

        // Clear and redraw GO button
        Button_Go.ClearButtonArea();
        Button_Go.DrawButton();
        Button_Go.DrawButtonText();
}

// Was it RUN you said ?
if (Button_Run.Pointer_in_area())
{
    // Just run through the protocol

    Button_Run.Do_click();
    line.ClearTextArea();
    line.DisplayText("Selected Run of the protocol");
    if (num_nodes == 3)
    {
        // Different for 3 principals
        for(j=0; j < textInstance→NumMessages(); j++)
        {
            sprintf(step_text,
            "Full Run option :  Step %d", j+1);
            line.ClearTextArea();
```

```
                line.DisplayText(step_text);
                msgStruct = textInstance→NextMessage();
                if ((!strcmp(msgStruct→sender, nodes[0]))
                    && (!strcmp(msgStruct→receiver, nodes[1])))
                    MessageXmitDisplay(&program_window,
                        arc_AB, msgStruct→message, right);
                if ((!strcmp(msgStruct→receiver, nodes[0]))
                    && (!strcmp(msgStruct→sender, nodes[1])))
                    MessageXmitDisplay(&program_window,
                        arc_AB, msgStruct→message, left);
                if ((!strcmp(msgStruct→sender, nodes[1]))
                    && (!strcmp(msgStruct→receiver, nodes[2])))
                    MessageXmitDisplay(&program_window,
                        arc_SB, msgStruct→message, right);
                if ((!strcmp(msgStruct→receiver, nodes[1]))
                    && (!strcmp(msgStruct→sender, nodes[2])))
                    MessageXmitDisplay(&program_window,
                        arc_SB, msgStruct→message, left);
                if ((!strcmp(msgStruct→sender, nodes[2]))
                    && (!strcmp(msgStruct→receiver, nodes[0])))
                    MessageXmitDisplay(&program_window,
                        arc_AS, msgStruct→message, right);
                if ((!strcmp(msgStruct→receiver, nodes[2]))
                    && (!strcmp(msgStruct→sender, nodes[0])))
                    MessageXmitDisplay(&program_window,
                        arc_AS, msgStruct→message, left);

                textInstance→increment_ctr();

            }
        }
        else
        {
            // Only 2 principals A and B
            for (j=0; j < textInstance→NumMessages(); j++)
            {
                msgStruct = textInstance→NextMessage();
                if ((!strcmp(msgStruct→sender, nodes[0]))
                    && (!strcmp(msgStruct→receiver, nodes[1])))
                    MessageXmitDisplay(&program_window,
                        arc_AB, msgStruct→message, right);
                if ((!strcmp(msgStruct→receiver, nodes[0]))
                    && (!strcmp(msgStruct→sender, nodes[1])))
                    MessageXmitDisplay(&program_window,
                        arc_AB, msgStruct→message, left);
                textInstance→increment_ctr();
            }
        }
        Button_Run.ClearButtonArea();
        Button_Run.DrawButton();
        Button_Run.DrawButtonText();
    }


    // Did you click on a beliefs subwindow, by any
    // chance ?
    if (report.xbutton.window ==
```

```
                    node_A_bw[beliefs_count]→Window_id())
                {
                    ret_str =
                    node_A_bw[beliefs_count]→Handle_click(
                        report.xbutton.button);
                    if (ret_str ≠ NULL)
                    {
                        if (!proofMode)
                        // Display proofs now..
                        proofsWindow = new BeliefsWindow(tdisplay,
                                    DefaultRootWindow(tdisplay),
                                    argc, argv, gc, "Proofs", 50, 50,
                                    400, 300, 1,
                                    BlackPixel(tdisplay,
                                    DefaultScreen(tdisplay)),
                                    WhitePixel(tdisplay,
                                    DefaultScreen(tdisplay)),
                                    common_font);
                        DisplayProofs(proofsWindow, ret_str, proofMode,
                                    argc, argv);

                        proofMode = True;
                        sleep(5);

                    }
                    else
                    {
                        // Improper mouse click
                        line.ClearTextArea();
                        line.DisplayText(
                        "Click on belief line beginning with the
belief");
                    }

                }
                if (report.xbutton.window ==
                    node_B_bw[beliefs_count]→Window_id())
                {
                    ret_str =
                    node_B_bw[beliefs_count]→Handle_click(
                        report.xbutton.button);
                    if (ret_str ≠ NULL)
                    {

                        if (!proofMode)
                        // Display proofs now..
                        proofsWindow = new BeliefsWindow(tdisplay,
                                    DefaultRootWindow(tdisplay),
                                    argc, argv, gc, "Proofs", 50, 50,
                                    400, 300, 1,
                                    BlackPixel(tdisplay,
                                    DefaultScreen(tdisplay)),
                                    WhitePixel(tdisplay,
                                    DefaultScreen(tdisplay)),
                                    common_font);
                        DisplayProofs(proofsWindow, ret_str, proofMode,
                                    argc, argv);
```

```
            proofMode = True;
        }
    }

    // Did you click on S's beliefs
    if (num_nodes == 3)
    if (report.xbutton.window ==
        node_S_bw[beliefs_count]→Window_id())
    {
        ret_str =
        node_S_bw[beliefs_count]→Handle_click(
                report.xbutton.button);
        if (ret_str ≠ NULL)
        {

            if (!proofMode)
            // Display proofs now..
            proofsWindow = new BeliefsWindow(tdisplay,
                                DefaultRootWindow(tdisplay),
                                argc, argv, gc, "Proofs", 50, 50,
                                400, 300, 1,
                                BlackPixel(tdisplay,
                                DefaultScreen(tdisplay)),
                                WhitePixel(tdisplay,
                                DefaultScreen(tdisplay)),
                                common_font);
            DisplayProofs(proofsWindow, ret_str, proofMode,
                                argc, argv);
            proofMode = True;
        }
    }

    // Did you click on Re-editing
    if (Button_Reedit.Pointer_in_area())
    {
        // Update status line first
        line.ClearTextArea();
        line.DisplayText("Selected Re-edit of beliefs");

        // Set re-edit mode to true
        reedit = True;

        // Build command and execute it for retaining
        // original assumptions file
        strcpy(assfile, "assump_2");

        // Delete any previous duplicate files
        unlink(assfile);

        sprintf(cpstr,"cp %s %s \n", argv[2],
                assfile);

        system(cpstr);

        // Re-edit assumptions, call handler for it
        program_window.Reedit_handler(assfile);
```

```
                    // Reset some variables
                    displayed_all_a = displayed_all_b = displayed_all_s
                        = False;


                    entered_reedit = True;
                }
                break;

case Expose:
                // Draw main window, status line, buttons and diagram
                program_window.DrawWindow();
                line.DrawStatusLine();
                line.DisplayText(sl_text);
                Button_Go.DrawButton();
                Button_Go.DrawButtonText();
                Button_Run.DrawButton();
                Button_Run.DrawButtonText();
                Button_Reedit.DrawButton();
                Button_Reedit.DrawButtonText();
                Button_Quit.DrawButton();
                Button_Quit.DrawButtonText();
                node_A.DrawNode();
                node_A.DrawNodeName();
                node_B.DrawNode();
                node_B.DrawNodeName();
                XFlush(tdisplay);
                if (num_nodes == 3)
                {
                    node_S→DrawNode();
                    node_S→DrawNodeName();
                    arc_AS→DrawArc();
                    arc_SB→DrawArc();
                }
                arc_AB→DrawArc();

                if (reedit)
                {
                    // If re-editing done, got to redraw new
                    // beliefs subwindows
                    if (beliefs_count > 0)
                    {
                        // If reediting is done, also display previous
                        // beliefs
                        if (report.xexpose.window ==
                            node_A_bw[beliefs_count-1]→Window_id())
                        {
                            node_A_bw[beliefs_count-1]→DrawItself();
                            node_A_bw[beliefs_count-1]→Init_y_offset();
                            node_A_bw[beliefs_count-1]→DisplayBeliefs(
                                retain);
                        }

                        if (report.xexpose.window ==
                            node_B_bw[beliefs_count-1]→Window_id())
                        {
                            node_B_bw[beliefs_count-1]→DrawItself();
```

```
                node_B_bw[beliefs_count-1]→Init_y_offset();
                node_B_bw[beliefs_count-1]→DisplayBeliefs(
                    retain);
            }

            if (num_nodes == 3)
                if (report.xexpose.window ==
                    node_S_bw[beliefs_count-1]→Window_id())
            {
                node_S_bw[beliefs_count-1]→DrawItself();
                node_S_bw[beliefs_count-1]→Init_y_offset();
                node_S_bw[beliefs_count-1]→DisplayBeliefs(
                    retain);
            }
        }
    }

    if (report.xexpose.window
        == node_A_bw[beliefs_count]→Window_id())
    {
        node_A_bw[beliefs_count]→DrawItself();
        node_A_bw[beliefs_count]→Init_y_offset();
        if (displayed_all_a)
            node_A_bw[beliefs_count]→DisplayBeliefs(retain);
        else
        {
            node_A_bw[beliefs_count]→DisplayBeliefs(all);
            displayed_all_a = True;
        }
    }

    if (report.xexpose.window ==
        node_B_bw[beliefs_count]→Window_id())
    {
        node_B_bw[beliefs_count]→DrawItself();
        node_B_bw[beliefs_count]→Init_y_offset();
        if (displayed_all_b)
            node_B_bw[beliefs_count]→DisplayBeliefs(retain);
        else
        {
            node_B_bw[beliefs_count]→DisplayBeliefs(all);
            displayed_all_b = True;
        }
    }

    if (num_nodes == 3)
        if (report.xexpose.window ==
            node_S_bw[beliefs_count]→Window_id())
    {
        node_S_bw[beliefs_count]→DrawItself();
        node_S_bw[beliefs_count]→Init_y_offset();
        if (displayed_all_s)
            node_S_bw[beliefs_count]→DisplayBeliefs(retain);
        else
        {
            node_S_bw[beliefs_count]→DisplayBeliefs(all);
            displayed_all_s = True;
```

```
                    }
                }

                // If proofs are being displayed, redisplay them
                if (proofMode)
                    if (report.xexpose.window ==
                        proofsWindow→Window_id())
                    {
                        proofsWindow→DrawItself();
                        proofsWindow→Init_y_offset();
                        proofsWindow→DisplayBeliefs(retain);
                    }
                XFlush(tdisplay);
                break;
            }
        }
}

void
MessageXmitDisplay(MainWindow *progWin, Arc *theArc, char *text,
                   arrowDir direction)
{
    Text *SampleText, *SampleText2;
    short len;
    int posx, posy;
    char tempstr1[40], tempstr2[40];
    Bool wrapped, super;

    super = wrapped = False;

    if (theArc == arc_AB)
    {
        // Coords for arc AB
        posx = 300;
        posy = 450;
    }

    if (theArc == arc_AS)
    {
        // Coords for arc AS
        posx = 50;
        posy = 300;
        len = strlen(text);

        // If text is really long then just display it below the
        // diagram
        if (len > SUPER_TEXT_LENGTH)
        {
            posx = 50;
            posy = 450;
            super = True;
        }
        else
        // Check if text overshoots beyond some length
        if (len > MAX_TEXT_LENGTH)
        {
            wrapped = True;
```

```
            strpartcpy(tempstr2, text, MAX_TEXT_LENGTH, len);
            strpartcpy(tempstr1, text, 0, MAX_TEXT_LENGTH - 1);
        }
    }

    if (theArc == arc_SB)
    {
        // Coords for arc SB
        posx = 450;
        posy = 300;
        len = strlen(text);

        // If text is really long then just display it below the
        // diagram
        if (len > SUPER_TEXT_LENGTH)
        {
            posx = 50;
            posy = 450;
            super = True;
        }
        else
        // Check if text overshoots beyond some length
        if (len > MAX_TEXT_LENGTH)
        {
            wrapped = True;
            strpartcpy(tempstr2, text, MAX_TEXT_LENGTH, len);
            strpartcpy(tempstr1, text, 0, MAX_TEXT_LENGTH - 1);
        }
    }

    if (theArc == arc_AB)
    {
        // Just a simple arc AB
        SampleText = new Text(tdisplay, progWin->base_window(), gc,
                                common_font, posx, posy, text);
        SampleText->DrawXCentredText();
    }
    else
    {
        if (!wrapped)
        {
            // Just draw one line of text
            SampleText = new Text(tdisplay, progWin->base_window(),
                                    gc, common_font, posx, posy, text);
            if (super)
                SampleText->DrawXCentredText();
            else
                SampleText->DrawText();
        }
        else
        {
            // Two instances of Text for two lines
            SampleText = new Text(tdisplay, progWin->base_window(), gc,
                                common_font, posx, posy, tempstr1);

            SampleText->DrawText();
```

```
            SampleText2 = new Text(tdisplay, progWin→base_window(), gc,
                                                common_font, posx, posy +
                                                TEXT_INCREMENT_Y, tempstr2);
            SampleText2→DrawText();
        }
    }

    theArc→HighlightArc(direction);
    sleep(5);
    theArc→UnHighlightArc(direction);
    sleep(1);
    SampleText→ClearText();
    delete SampleText;
    if (wrapped)
    {
        SampleText2→ClearText();
        delete SampleText2;
    }
}

void
MessageXmitHighlight(MainWindow *progWin, Arc *theArc, Text *SampleText,
                        TextInput *theTextInstance, arrowDir *dirntable,
                        unsigned short flag)
{
    static short count = 0;

    if (flag == HIGHLIGHT)
    {
        SampleText→DrawXCentredText();
        theArc→HighlightArc(dirntable[count]);
    }
    else if (flag == UNHIGHLIGHT)
    {
            theArc→UnHighlightArc(dirntable[count++]);
            SampleText→ClearText();
            delete SampleText;
    }
}

void
DisplayProofs(BeliefsWindow *proofsWindow, char *string, Bool Mode, int argc,
                char *argv[])
{
    Gny *gnyObject;
    char tstr[20];
    unsigned short i;

    // Actually draw the proofs window
    proofsWindow→Init_y_offset();
    proofsWindow→DrawItself();

    gnyObject = new Gny("T", assfile);

    // Build file name
    sprintf(tstr,"m_%d",Go_loop_count);
```

```
    // Get proofs into object
    gnyObject→Gny_get_beliefs_and_proofs(tstr, PROOFS, string);

    if (Mode)
        proofsWindow→DestroyBeliefs();

    // Load proofs
    for (i=0;i<gnyObject→Gny_num_beliefs(); i++)
        proofsWindow→LoadBeliefs(gnyObject→Gny_return_current_belief());


    // Display them finally
    proofsWindow→DrawItself();
    proofsWindow→DisplayBeliefs(all);

    //delete gny object
    delete gnyObject;
}

void
strpartcpy(char *string1, char *string2, short start,
                        short end)
{
    short i, j;

    // Simple transfer loop
    for (i=start,j=0;i<(start + end);i++,j++)
            string1[j] = string2[i];
}


//////////////////////////////////////////////////////////
//
// File : DisplayBase.h
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the global declarations
// and class specification for class
// DisplayBase.
//
//////////////////////////////////////////////////////////

#define _DISPLAYBASE_

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <stream.h>
#include <fstream.h>

// Global # defines
```

```
#define TRUE 1
#define FALSE 0
#define BASE_WD 800
#define BASE_HT 600
#define BASE_X 200
#define BASE_Y 200
#define NODE_WIDTH 60
#define DISPLAY_CENTRE_X 400
#define BUTTON_WD 100
#define BUTTON_HT 50
#define CIRCLE_START 0
#define CIRCLE_END 360*64
#define SCROLLBAR_WIDTH 17
#define DEFAULT_TEXT_HEIGHT 30
#define LINELENGTH 140
#define BELIEFS 1
#define PROOFS 2
#define ENTITY_LENGTH 5
#define MAX_NODES 5
#define FLAG "flag"
#define FACT "fact"
#define POSSESSES "possesses("
#define TOLD "told("
#define CONVEYED "conveyed("
#define BELIEVES "believes("
#define REASON ", reason("
#define REASON_SP ",reason("
#define EXT "ext("
#define STAR "star("
#define COMMA ","
#define GNY_INPUTFILE "temp"
#define GNY_ASSUMPTIONSFILE "assumptions"

// Typedef for beliefs display
typedef enum
{
    up,
    down,
    all,
    retain
} Direction;

// Typedef for arc direction
typedef enum
{
    right,
    left
} arrowDir;

// Typedef for messages structure
typedef struct
{
    char *sender;
    char *receiver;
    char *message;
} MessageStruct;
```

```
class DisplayBase
{
    public:
    virtual void Do_click() {return; };
    virtual Pointer_in_area() const;

    protected:
    Display *fdisplay;
    Window fcurrent_window;
    GC fgc;
};


//////////////////////////////////////////////////////////
//
// File : DisplayBase.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the member
// functions for the class DisplayBase
//
//
//////////////////////////////////////////////////////////

#include "DisplayBase.h"

// Default Pointer_in_area in base class
Bool
DisplayBase::Pointer_in_area() const
{
    Window root_win, child_win;
    int root_x, root_y, win_x, win_y;
    unsigned int mask;

    // Return true if in my area
    if (XQueryPointer(fdisplay, fcurrent_window, &root_win, &child_win, &root_x,
                      &root_y, &win_x, &win_y, &mask))
        return True;

    return False;
}


//////////////////////////////////////////////////////////
//
// File : Gny.h
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the global declarations
// and class specification for class Gny.
//
```

```
//
//////////////////////////////////////////////////////

#ifndef _DISPLAYBASE_
#include "DisplayBase.h"
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ctype.h>

// Class Gny # defines
#define OUTPUTFILENAME "xsb_output"
#define NEWLINE "\n"
#define MEMORY_SIZE 2000
#define MAX_BELIEFS 100
#define MAX_PROOFS 100
#define BELIEFSLENGTH 550
#define PROOFSLENGTH 570

class Gny
{
    public:
    Gny(char *str, char *filename);
    ~Gny() { delete fbeliefs; delete fname_node; unlink(OUTPUTFILENAME); };

    void Gny_get_beliefs_and_proofs(char *input_filename);
    void Gny_get_beliefs_and_proofs(char *input_filename,
            unsigned short flag, char *input_text);
    char* Gny_return_current_belief();
    unsigned short Gny_num_beliefs() { return fnum_beliefs; };
    void Gny_parse_file(char *filename, unsigned short flag);

    private:

    char *trim_belief_line(char *line);

    char *fname_node;
    char *fassumptionsfile;
    char *fbeliefs[MAX_BELIEFS];
    unsigned short fnum_beliefs;
    unsigned short fpointer_b;
};

//////////////////////////////////////////////////////
//
// File : Gny.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains member and other
// functions for class Gny.
//
```

```
//
//////////////////////////////////////////////////////
#include "Gny.h"

inline Bool
MatchStrings(char *str1, char *str2)
{
    unsigned short len;
    unsigned short cnt;

    len = strlen(str2);

    for(cnt = 0;cnt < len;cnt++)
        if (str1[cnt] ≠ str2[cnt])
            return FALSE;

    return TRUE;
}

inline char *
strip_facts(char *string)
{
    char *dum_str;
    char *end_ptr;
    char *temp_str;

    if (!strstr(string, FACT))
        return string;

    temp_str = new char[strlen(string) + 1];
    strcpy(temp_str, string);

    dum_str = strchr(temp_str, ',');

    if (dum_str ≠ NULL)
    {
        dum_str++;
        while (isspace(*dum_str))
            dum_str++;
    }

    temp_str = dum_str;

    end_ptr = strrchr(temp_str,')');

    *end_ptr = NULL;

    return temp_str;
}

inline char *
CheckString(char *inputString, char *nameNode, unsigned short flag)
{
    char *local_str;
    Bool local_str_tag;
    char tempstr[20];
    char *check_str;
```

```
char *copy_str;

local_str_tag = FALSE;
local_str = NULL;

if (!strstr(inputString, POSSESSES) && !strstr(inputString, TOLD) &&
    !strstr(inputString, CONVEYED) && !strstr(inputString,
    BELIEVES))
    return NULL;

copy_str = strip_facts(inputString);
if (flag == PROOFS)
{
    // Filter out necessary staements for proofs
    local_str = inputString;
    while (!isdigit(*local_str) && (*local_str ≠ NULL))
            local_str++;

    if ((isdigit(*local_str) && (*(local_str + 1) == '.'))
            && (strstr(local_str, POSSESSES) ||
                strstr(local_str, TOLD) ||
                strstr(local_str, CONVEYED) ||
                strstr(local_str, BELIEVES)))
    {
        local_str[strlen(local_str)-1] = NULL;
        return local_str;
    }

    if (*local_str == NULL)
        return NULL;
}

// For beliefs...
memset(tempstr, 0, 20);
sprintf(tempstr,"%s%s",POSSESSES,nameNode);
local_str_tag = MatchStrings(copy_str, tempstr);
if (!local_str_tag )
{
    memset(tempstr, 0, 20);
    sprintf(tempstr,"%s%s",TOLD,nameNode);
    local_str_tag = MatchStrings(copy_str, tempstr);
}
if (!local_str_tag )
{
    memset(tempstr, 0, 20);
    sprintf(tempstr,"%s%s",CONVEYED,nameNode);
    local_str_tag = MatchStrings(copy_str, tempstr);
}

if (!local_str_tag )
{
    memset(tempstr, 0, 20);
    sprintf(tempstr,"%s%s",BELIEVES,nameNode);
    local_str_tag = MatchStrings(copy_str, tempstr);
}

if (local_str_tag)
```

```
        return inputString;
    else
        return NULL;

}


Gny::Gny(char *str, char *filename)
{
    fnum_beliefs = 0;
    fpointer_b = 0;

    // Allocate space for the name of the node and copy it
    fname_node = new char[strlen(str) + 1];
    strcpy(fname_node, str);

    // Set assumptions filename
    fassumptionsfile = new char[strlen(filename) + 1];
    strcpy(fassumptionsfile, filename);

}

char *
Gny::trim_belief_line(char *line)
{
    char *local_str, *t_str;
    char ano_str[20];

    local_str = strstr(line, POSSESSES);
    if (local_str == NULL)
        local_str = strstr(line, TOLD);
    if (local_str == NULL)
        local_str = strstr(line, CONVEYED);
    if (local_str == NULL)
        local_str = strstr(line, BELIEVES);

    if (local_str ≠ NULL)
    {
        // Get pointer to actual belief in line
        if (local_str = strstr(line, COMMA))
            local_str++;

        t_str = strstr(local_str,REASON);
        if (t_str == NULL)
            t_str = strstr(local_str, REASON_SP);
        if (t_str ≠ NULL)
        {
            *t_str = NULL;
            return local_str;
        }
        else
            return NULL;
    }
}

void
Gny::Gny_get_beliefs_and_proofs(char *input_filename, unsigned short flag,
```

```
                                        char *input_text)
{
    char temp_string[80], appendstring[40];
    char local_file1[] = "localmessagefile";
    char local_file2[] = "t_localmessagefile";
    short i;
    char commands[4][PROOFSLENGTH];
    FILE *fptr;
    char *t_str;

    // While debugging only
    unlink(local_file1);
    unlink(local_file2);

    // Initialize standard commands for xsb interaction
    strcpy(commands[0], "[basics,control,gny_mod].");
    sprintf(commands[1],"analyze(%s).",local_file1);
    if (flag == BELIEFS)
        strcpy(commands[2], "listing.");
    else
    if (flag == PROOFS)
    {
        t_str = trim_belief_line(input_text);
        sprintf(commands[2],"explain_proof(%s).",t_str);
    }

    // All operations on a local file
    sprintf(appendstring,"cp %s %s",input_filename, local_file1);

    // Execute command built above
    system(appendstring);

    // Append assumptions file to message file
    memset(appendstring,0, 40);
    sprintf(appendstring,"cat %s >> %s",fassumptionsfile,
            local_file1);
    system(appendstring);

    fptr = fopen(local_file2,"w+");

    for (i=0;i<3;i++)
    {
        fprintf(fptr,"%s \n",commands[i]);
    }

    fclose(fptr);

    // Run the GNY tool with the files created so far and store the
    // output in another file
    sprintf(temp_string, "xsb -m %d -i < %s > %s", MEMORY_SIZE,
            local_file2, OUTPUTFILENAME);

    // Execute the xsb process
    system(temp_string);

    Gny_parse_file(OUTPUTFILENAME, flag);
```

```
    // Delete the unwanted files
    unlink(local_file1);
    unlink(local_file2);
}

char *
Gny::Gny_return_current_belief()
{
    // Return current belief
    return fbeliefs[fpointer_b++];
}

void
Gny::Gny_parse_file(char *filename, unsigned short flag)
{
    FILE *fptr;
    char readstring[BELIEFSLENGTH];
    short count;
    char *t_string;

    // Open GNY output file
    fptr = fopen(filename, "r+");
    if (fptr == NULL) return;

    count = fnum_beliefs;

    // Load valid beliefs into GNY object
    while (!feof(fptr) && (count < 199))
    {
        memset(readstring, 0, BELIEFSLENGTH);
        fgets(readstring, BELIEFSLENGTH, fptr);
        t_string = CheckString(readstring, fname_node, flag);
        if (t_string ≠ NULL)
        {
            fbeliefs[count] = new char[strlen(t_string) + 1];
            strcpy(fbeliefs[count++], t_string);
            fnum_beliefs++;
        }
    }
    fclose(fptr);

}


//////////////////////////////////////////////////////////
//
// File : MainWindow.h
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the global declarations
// and class specification for the class
// MainWindow.
//
//////////////////////////////////////////////////////////
```

```
#ifndef _DISPLAYBASE_
#include "DisplayBase.h"
#endif

#define COMMAND_LENGTH 25

// MainWindow globals
XSizeHints xsh;
XWMHints wm_hints;
XClassHint class_hints;
XTextProperty windowName, iconName;

class MainWindow : public DisplayBase
{
    public:
    MainWindow(int margc, char *margv[],
                Window parent_wid, GC gc, char *window_name, char *icon_name,
                Display *display, short width, short height,
                short start_x, short start_y);

    ~MainWindow();

    void DrawWindow() const;
    void Reedit_handler(char *assump_file) const;
    Window base_window() const;

    private:
    short fstart_x;
    short fstart_y;
    short fsize_x;
    short fsize_y;
    char *fwin_name;
    char *ficon_name;

};


//////////////////////////////////////////////////////
//
// File : MainWindow.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains member functions
// for the class MainWindow
//
//
//////////////////////////////////////////////////////

#include "MainWindow.h"

MainWindow::MainWindow(int margc, char *margv[],
                        Window parent_wid, GC gc, char *window_name,
                        char *icon_name,
```

```
                        Display *display, short width, short height,
                        short start_x, short start_y )
{

    // Set class variables
    fdisplay = display;
    fgc = gc;
    fwin_name = window_name;
    ficon_name = icon_name;
    fstart_x = start_x;
    fstart_y = start_y;
    fsize_x = width;
    fsize_y = height;

    // Create the base window first
    fcurrent_window = XCreateSimpleWindow(fdisplay, parent_wid,
                                          start_x, start_y, width, height, 0,
                                          BlackPixel(fdisplay,
                                          DefaultScreen(fdisplay)),
                                          WhitePixel(fdisplay,
                                          DefaultScreen(fdisplay)));


    xsh.flags = (PPosition | PSize);
    xsh.height = height;
    xsh.width = width;
    xsh.x = start_x;
    xsh.y = start_y;

    wm_hints.initial_state = NormalState;
    wm_hints.flags = StateHint;

    class_hints.res_name = fwin_name;

    // Window name to window manager
    if (XStringListToTextProperty(&fwin_name, 1, &windowName) == 0)
    {
        cerr << fwin_name <<
            " :  structure allocation for windowName failed.  \n";
        exit (-1);
    }

    if (XStringListToTextProperty(&fwin_name, 1, &iconName) == 0)
    {
        cerr << fwin_name <<
            " :  structure allocation for iconName failed.  \n";
        exit (-1);
    }

    // Hints to window manager
    XSetWMProperties(fdisplay, fcurrent_window, &windowName, &iconName,
                     margv, margc, &xsh, &wm_hints, &class_hints);

}

MainWindow::~MainWindow()
{
```

```
    // Destroy base window
    XDestroyWindow(fdisplay, fcurrent_window);

}

void
MainWindow::DrawWindow() const
{
    // Set mask for Expose, ButtonPress and KeyPress events
    XSelectInput(fdisplay, fcurrent_window, ExposureMask | ButtonPressMask
                        | KeyPressMask );

    // Map base window
    XMapWindow(fdisplay, fcurrent_window);

    //Clear the window
    XClearWindow(fdisplay, fcurrent_window);

    XFlush(fdisplay);
}

Window
MainWindow::base_window() const
{
    // Return window id of main window
    return fcurrent_window;
}

void
MainWindow::Reedit_handler(char *assump_file) const
{
    char *editor;
    char command[COMMAND_LENGTH];

    // Get environment variable EDITOR
    editor = getenv("EDITOR");

    if (editor == NULL)
    {
        // Tough.., use vi
        editor = new char[strlen("vi") + 1];
        strcpy(editor, "vi");
    }

    // Build the command for xterm
    sprintf(command,"xterm -e %s %s",editor, assump_file);

    // Execute the xterm with editor now
    system(command);

}

//////////////////////////////////////////////////////
//
// File : Node.h
//
//
```

```
// Author : Viswanathan Narain
//
//
// Description : This file contains the class specification
// for class Node.
//
//
////////////////////////////////////////////////////////////

#ifndef _DISPLAYBASE_
#include "DisplayBase.h"
#endif

class Node : public DisplayBase
{
    public:
    Node(Display *display, Window parent_win, GC gc, char *fontname,
            char *node_name, int pos_x, int pos_y, unsigned int width,
            unsigned int height, unsigned int angle1, unsigned int angle2);

    void DrawNode() const;
    void DrawNodeName() const;
    Bool Pointer_in_area() const;

    private:
    int fpos_x;
    int fpos_y;
    int fwidth;
    int fheight;
    int fangle1;
    int fangle2;
    char *fnode_name;
};

////////////////////////////////////////////////////////////
//
// File : Node.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains member and other
// functions for class Node.
//
//
////////////////////////////////////////////////////////////

#include "Node.h"

inline Bool
In_circular_area(int centre_x, int centre_y, int radius, int pt_x, int pt_y)
{
    // Are coords within circular area of node
    if ((((pt_x - centre_x)*(pt_x - centre_x)) +
        ((pt_y - centre_y) * (pt_y - centre_y)))
            <= (radius * radius))
```

```
            return True;

    // No, return false
    return False;
}

Node::Node(Display *display, Window parent_win, GC gc, char *fontname,
            char *node_name, int pos_x, int pos_y, unsigned int width,
            unsigned int height, unsigned int angle1, unsigned int angle2)
{
    XFontStruct *font_ptr;

    fdisplay = display;
    fcurrent_window = parent_win;
    fgc = gc;
    fnode_name = new char[strlen(node_name) + 1];
    strcpy(fnode_name, node_name);
    fpos_x = pos_x;
    fpos_y = pos_y;
    fwidth = width;
    fheight = height;
    fangle1 = angle1;
    fangle2 = angle2;

    // Load font specified in fontname
    if ((font_ptr = XLoadQueryFont(fdisplay, fontname)) == NULL)
    {
        cerr << "Error loading font :" << fontname << "\n";
        return;
    }
    XSetFont(fdisplay, fgc, font_ptr->fid);
}

void
Node::DrawNode() const
{
    // Draw the node
    XDrawArc(fdisplay, fcurrent_window, fgc, fpos_x, fpos_y, fwidth, fheight,
                fangle1, fangle2);

    // Flush X display
    XFlush(fdisplay);
}

void
Node::DrawNodeName() const
{
    int offset_x, offset_y;

    offset_x = fwidth/2;
    offset_y = fheight/2;

    // Draw principal's name within node
    XDrawImageString(fdisplay, fcurrent_window, fgc, fpos_x + offset_x,
                        fpos_y + offset_y, fnode_name, 1);
    XFlush(fdisplay);
}
```

```
Bool
Node::Pointer_in_area() const
{
    Window root_win, child_win;
    int root_x, root_y, win_x, win_y;
    unsigned int mask;
    int centre_x, centre_y;

    centre_x = fpos_x + (fwidth/2);
    centre_y = fpos_y + (fheight/2);

    // Got to check if you clicked in my circle
    if (XQueryPointer(fdisplay, fcurrent_window, &root_win, &child_win,
        &root_x, &root_y, &win_x, &win_y, &mask))
        if (In_circular_area(centre_x, centre_y, fwidth/2, win_x, win_y))
            return True;

    return False;
}


//////////////////////////////////////////////////////////
//
// File : ScrollBar.h
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the global declarations
// and class specification for the class
// ScrollBar.
//
//////////////////////////////////////////////////////////

#define _SCROLLBAR_

#ifndef _DISPLAYBASE_
#include "DisplayBase.h"
#endif

#ifndef _BUTTON_
#include "Button.h"
#endif

// y direction offset
#define INCREMENT_Y 15

class ScrollBar : public Button
{
    public:
    ScrollBar(Display *display, Window parent_win, GC gc,
                int left_x, int left_y, unsigned int width,
                unsigned int height, unsigned int offset_x,
                unsigned int offset_y);
```

```
    void Handle_click(unsigned int button);

    private:
    void Scroller();

    Direction fdir;
    unsigned int foffset_y;
    unsigned int current_scroll_x;
    unsigned int current_scroll_y;
    unsigned int width_text_area;
    unsigned int height_text_area;
};

//////////////////////////////////////////////////////
//
// File : ScrollBar.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains member functions for
// the class ScrollBar.
//
//
//////////////////////////////////////////////////////

#include "ScrollBar.h"

ScrollBar::ScrollBar(Display *display, Window parent_win, GC gc,
                     int left_x, int left_y, unsigned int width,
                     unsigned int height, unsigned int offset_x,
                     unsigned int offset_y) : Button(display,
                     parent_win, gc, left_x, left_y, width, height,
                     NULL, NULL)
{
    fdisplay = display;
    fcurrent_window = parent_win;
    fgc = gc;
    fleft_x = left_x;
    fleft_y = left_y;
    fwidth = width;
    fheight = height;
    foffset_y = offset_y;

    current_scroll_x = fleft_x + offset_x;
    current_scroll_y = fleft_y;

    // Width of enclosing text area less by size of scrollbar
    width_text_area = 400 - offset_x;

    // Height same
    height_text_area = fheight;
}

void
ScrollBar::Handle_click(unsigned int button)
```

```
{

    switch(button)
    {
    case 1:
            // Scroll down
            fdir = down;
            Scroller();
            break;

    case 2:
            // Scroll up
            fdir = up;
            Scroller();
            break;
    }
}

void
ScrollBar::Scroller()
{
    Bool exposures = True;

    switch(fdir)
    {
    case up:
            // Clear one line at the top
            XClearArea(fdisplay, fcurrent_window,
                    current_scroll_x, height_text_area - foffset_y,
                    width_text_area,
                    foffset_y , exposures);
            XFlush(fdisplay);
            current_scroll_y -= foffset_y;

            // Copy bottom area to top
            XCopyArea(fdisplay, fcurrent_window, fcurrent_window,
                    fgc, current_scroll_x, current_scroll_y,
                    width_text_area, height_text_area,
                    current_scroll_x, current_scroll_y +
                    foffset_y);

            // Clear small line portion below
            XClearArea(fdisplay, fcurrent_window,
                    current_scroll_x, current_scroll_y, width_text_area,
                    foffset_y , exposures);
            XFlush(fdisplay);
            break;

    case down:
                // Clear one bottommost line
                XClearArea(fdisplay, fcurrent_window,
                        current_scroll_x, current_scroll_y,
                        width_text_area ,
                        foffset_y , exposures);
                current_scroll_y += foffset_y;

                // Copy top portion below
```

```
                    XCopyArea(fdisplay, fcurrent_window, fcurrent_window,
                              fgc, current_scroll_x, current_scroll_y,
                              width_text_area, height_text_area,
                              current_scroll_x, current_scroll_y -
                              foffset_y);

                    // Clear topmost line
                    XClearArea(fdisplay, fcurrent_window,
                              current_scroll_x, height_text_area -
                              foffset_y, width_text_area,
                              foffset_y , exposures);
                    XFlush(fdisplay);
                    break;
    }
}

////////////////////////////////////////////////////////
//
// File : StatusLine.h
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the global declarations
// and class specification for the class
// StatusLine.
//
////////////////////////////////////////////////////////

#ifndef _DISPLAYBASE_
#include "DisplayBase.h"
#endif

// y direction offset
#define Y_OFFSET 18

class StatusLine : public DisplayBase
{
    public:
    StatusLine(Display *display, Window parent_win, GC gc, char *fontname,
               int pos_x, int pos_y, int width, int height);

    void DrawStatusLine() const;
    void DisplayText(char *text) const;
    void ClearTextArea() const;

    private:
    XFontStruct *fp_font;
    int fpos_x;
    int fpos_y;
    int fwidth;
    int fheight;
};

////////////////////////////////////////////////////////
//
```

```
// File : StatusLine.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the member functions
// for class StatusLine.
//
//
////////////////////////////////////////////////////////

#include "StatusLine.h"

StatusLine::StatusLine(Display *display, Window parent_win, GC gc,
    char *fontname, int pos_x, int pos_y, int width, int height)
{

    fdisplay = display;
    fcurrent_window = parent_win;
    fgc = gc;
    fpos_x = pos_x;
    fpos_y = pos_y;
    fwidth = width;
    fheight = height;

    // Load font for this object
    if ((fp_font = XLoadQueryFont(fdisplay, fontname)) == NULL)
    {
        cerr << "Error in loading font :" << fontname << "\n";
        return;
    }
    XSetFont(fdisplay, fgc, fp_font->fid);
}

void
StatusLine::DrawStatusLine() const
{
    unsigned long foreground, background;

    foreground = 1;
    background = 0;
    XSetForeground(fdisplay, fgc, foreground);
    XSetBackground(fdisplay, fgc, background);

    // Draw a box filled with background colour
    XFillRectangle(fdisplay, fcurrent_window, fgc, fpos_x, fpos_y,
                   fwidth, fheight);
}

void
StatusLine::DisplayText(char *text) const
{
    int x_position;
    short text_len;
    int text_width;
```

```
    text_len = strlen(text);

    // Get width of each character in this font to calculate size
    // of entire string
    text_width = XTextWidth(fp_font, text, text_len);
    x_position = fwidth/2 - text_width/2;

    // Now draw the text on the box, centred
    XDrawImageString(fdisplay, fcurrent_window, fgc, x_position,
                            fpos_y + Y_OFFSET, text, text_len);
    XFlush(fdisplay);
}

void
StatusLine::ClearTextArea() const
{
    Bool exposures;

    exposures = True;

    // Get rid of the text
    XClearArea(fdisplay, fcurrent_window, fpos_x, fpos_y, fwidth,
                fheight, exposures);

    // Redraw the box
    XFillRectangle(fdisplay, fcurrent_window, fgc, fpos_x, fpos_y,
                        fwidth, fheight);
}

//////////////////////////////////////////////////////////////
//
// File : Text.h
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the class specification
// for class Text.
//
//
//////////////////////////////////////////////////////////////

#ifndef _DISPLAYBASE_
#include "DisplayBase.h"
#endif

class Text : public DisplayBase
{
    public:
    Text(Display *display, Window parent_win, GC gc, char *fontname, int pos_x,
            int pos_y, char *string);

    void DrawText() const;
    void DrawXCentredText();
    void ClearText() const;
```

```
    private:
    int fpos_x;
    int fpos_y;
    XFontStruct *fp_font;
    char *fstring;
    short fstring_size;
    short ftext_width;
    short ftext_height;
};

////////////////////////////////////////////////////////
//
// File : Text.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the member functions
// for class Text.
//
//
////////////////////////////////////////////////////////

#include "Text.h"

Text::Text(Display *display, Window parent_win, GC gc, char *fontname,
           int pos_x, int pos_y, char *string)
{
    fdisplay = display;
    fcurrent_window = parent_win;
    fgc = gc;
    fpos_x = pos_x;
    fpos_y = pos_y;

    // Store string in object
    fstring_size = strlen(string);
    fstring = new char[fstring_size];
    strcpy(fstring, string);

    if ((fp_font = XLoadQueryFont(fdisplay, fontname)) == NULL)
    {
        cerr << "Cannot load font :  %s\n" << fontname;
        cout.flush();
    }
    XSetFont(fdisplay, fgc, fp_font->fid);
    ftext_width = XTextWidth(fp_font, fstring, fstring_size);
    ftext_height = DEFAULT_TEXT_HEIGHT;
}

void
Text::DrawText() const
{
    // Draw the text now
    XDrawImageString(fdisplay, fcurrent_window, fgc, fpos_x, fpos_y,
                     fstring, fstring_size);
}
```

```
void
Text::DrawXCentredText()
{
    int x_position;

    // Get to the centre in the x direction
    fpos_x = BASE_WD/2 - ftext_width/2;

    DrawText();
}


void
Text::ClearText() const
{
    Bool exposures = False;

    // Clear the text
    XClearArea(fdisplay, fcurrent_window, fpos_x, fpos_y - ftext_height,
               ftext_width, ftext_height, exposures);

    // Flush X display
    XFlush(fdisplay);
}

//////////////////////////////////////////////////////////
//
// File : TextInput.h
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the global declarations
// and class specification for the class
// TextInput.
//
//////////////////////////////////////////////////////////

#ifndef _DISPLAYBASE_
#include "DisplayBase.h"
#endif

#include <ctype.h>

// TextInput # defines
#define MAX_MESSAGES 50
#define MAX_SENDERS 5
#define MAX_RECEIVERS 5
#define LINESIZE 100
#define NL 0x0a
#define MESSAGESFILE "voting"
#define MESSAGETAG "Messages"
#define ASSUMPTIONSTAG "Assumptions"
#define MAX_FORMULAE 10
```

```
// typedef for idealized messages structure
typedef struct Struct_idealized
{
    char *text;
    char formulae[10][550];
    unsigned short num_formulae;
} Struct_idealized;

class TextInput
{
    public:
    TextInput(char *filename, char *assumptions_filename);
    ~TextInput();
    void LoadMessages();
    MessageStruct *NextMessage();
    void Reinitialize_message_count() { current_message = 0; };
    unsigned short NumMessages() { return num_messages; };
    arrowDir *ReturnArrowsTable();
    void Transform_to_idealized();
    void Fill_formulae(unsigned short count);
    void Insert_stars_and_extensions();
    void Load_assumptions();
    unsigned short return_num_nodes() const;
    char *Return_sender_name();
    char *Return_receiver_name();
    void Dump_to_file();
    char *Return_msg_file();
    char *Return_t_msg_file();
    void increment_ctr() { current_message++; };

    private:
    void ParseMessage(char *message);

    char *fmessagesfile;
    char *fassumptionsfile;
    char *messages[MAX_MESSAGES];
    Struct_idealized idealized[MAX_MESSAGES];
    unsigned short current_message;
    unsigned short num_messages;
    char senders[MAX_SENDERS][ENTITY_LENGTH];
    char sender_assumptions[MAX_SENDERS][500];
    char receivers[MAX_RECEIVERS][ENTITY_LENGTH];
    char msg_files[MAX_MESSAGES][20];
    char t_msg_files[MAX_MESSAGES][20];
};

//////////////////////////////////////////////////////////
//
// File : TextInput.C
//
//
// Author : Viswanathan Narain
//
//
// Description : This file contains the member and other
// functions for class TextInput.
//
```

```
//
//////////////////////////////////////////////////////////

#include "TextInput.h"

inline Bool
ValidFormula(char *formula)
{
    // Check for validity of hashed, encrypted or decrypted
    // components
    if (strstr(formula, "h(") || strstr(formula, "encrypt(")
        || strstr(formula, "decrypt("))
        return True;
    else
        return False;
}

inline void
strip_newline(char *str)
{
    // This assumes that newline terminates a line
    while (*str)
    {
        if (*str == NL)
            *str = NULL;
        str++;
    }
}

inline Bool
IsStringValid(char *str)
{
    Bool Return = False;

    while (TRUE)
    {
        if (*str)
        {
            if (*str == '%')
            {
                // Forget this line, is a comment
                Return = False;
                return Return;
            }
            if (isalnum(*str))
                Return = True;
            str++;
        }
        else
            return Return;
    }
}

TextInput::TextInput(char *filename, char *assumptions_filename)
{
    short i,j;
```

```
    num_messages = 0;
    current_message = 0;

    // Name of protocol specification file
    fmessagesfile = new char[strlen(filename) + 1];
    strcpy(fmessagesfile, filename);

    // Name of initial assumptions file
    fassumptionsfile = new char[strlen(assumptions_filename) + 1];
    strcpy(fassumptionsfile, assumptions_filename);

    // Initialize senders
    for (i=0; i< MAX_SENDERS; i++)
        memset(senders[i], 0, ENTITY_LENGTH);

    // Initialize receivers
    for (i=0; i< MAX_RECEIVERS; i++)
        memset(receivers[i], 0, ENTITY_LENGTH);

    // Initialize idealized structure
    for (j=0; j < MAX_MESSAGES; j++)
        for (i=0; i < MAX_FORMULAE; i++)
        {
            memset(idealized[j].formulae[i], 0, 300);
            idealized[j].num_formulae = 0;
        }

    // Initialise assumptions table too..
    for (j=0; j < MAX_SENDERS; j++)
        memset(sender_assumptions[j], NULL, 250);
}

TextInput::~TextInput()
{
    short i;

    delete fmessagesfile;
    delete fassumptionsfile;

    // delete messages
    for (i=0; i < num_messages; i++)
        delete [] messages;

    // Delete all the temporary files
    for (i=0; i < num_messages; i++)
    {
        unlink(msg_files[i]);
        unlink(t_msg_files[i]);
    }
}

void
TextInput::LoadMessages()
{
    char temp_str[LINESIZE];
    Bool Read = False;
    FILE *fptr;
```

```
    if ( (fptr = fopen(fmessagesfile, "r+")) == NULL )
    {
        cerr << "Cannot open file :" << fmessagesfile <<
                "for input \n";
        exit (-1);
    }

    cout << "messgs file" << fmessagesfile <<"\n";
    cout.flush();

    fgets(temp_str, LINESIZE, fptr);
    strip_newline(temp_str);
    while (!feof(fptr))
    {
        // If input line is valid then
        if (IsStringValid(temp_str))
        {
            ParseMessage(temp_str);
            num_messages += 1;
        }
        fgets(temp_str, LINESIZE, fptr);
        strip_newline(temp_str);
    }

    cout << "recv 0 in Load msgs ->" << receivers[0] << "\n";
    cout.flush();
    // Set message counter back to first one
    current_message = 0;

    // Close the messages file
    fclose(fptr);

    // Load senders' assumptions
    Load_assumptions();

}

MessageStruct *
TextInput::NextMessage()
{
    MessageStruct *theMessageStruct;

    if (current_message < num_messages)
    {
        theMessageStruct = new MessageStruct[1];
        theMessageStruct->sender =
            new char[strlen(senders[current_message]) + 1];
        strcpy(theMessageStruct->sender, senders[current_message]);
        theMessageStruct->receiver =
            new char[strlen(receivers[current_message]) + 1];
        strcpy(theMessageStruct->receiver, receivers[current_message]);
        theMessageStruct->message =
            new char[strlen(messages[current_message]) + 1];
        strcpy(theMessageStruct->message, messages[current_message]);

        // Return the current message in this structure
```

```
        return theMessageStruct;
    }
    else
    {
        // Recursive, return first message again
        current_message = 0;
        return (theMessageStruct = NextMessage());
    }
}

arrowDir *
TextInput::ReturnArrowsTable()
{
    arrowDir *localarr;

    localarr = new arrowDir[num_messages];

    localarr[0] = right;
    localarr[1] = left;
    localarr[2] = right;

    return localarr;
}

void
TextInput::ParseMessage(char *message)
{
    char *message_part, entity_part[10];
    static unsigned short m_count = 0;
    Bool got_sender;
    short i, j, k;

    if (num_messages == 0)
        m_count = 0;

    memset(entity_part, 0, 10);

    // Point message to after message number at the beginning
    while (*message ≠ ')')
            message++;
    message++;

    got_sender = False;

    // Get text before ":" into entity part
    message_part = strchr(message, ':');
    strncpy(entity_part,message, abs(message - message_part));

    while (!isalnum(*message_part))
            message_part++;

    j = 0;
    k= 0;
    for(i=0;i<10;i++)
    {
        // Sender and receiver
        if (!got_sender && (entity_part[i] ≠ '-') &&
```

```
            (!isspace(entity_part[i])))
        {
            senders[m_count][j++] = entity_part[i];
        }
        if (got_sender && (entity_part[i] ≠ '-') &&
            (entity_part[i] ≠ '>') && (!isspace(entity_part[i])))
            receivers[m_count][k++] = entity_part[i];
        if (entity_part[i] == '-')
        {
            senders[m_count][j] = NULL;
            got_sender = True;
        }
    }
    receivers[m_count][k] = NULL;

    cout ≪ "Send ->" ≪ senders[m_count] ≪ " Recv ->" ≪ receivers[m_count] ≪
"\n";
    cout.flush();
    // Store message text into messages
    messages[m_count] = new char[strlen(message_part) + 1];
    strcpy(messages[m_count], message_part);

    m_count += 1;

    cout ≪ "recv 0 in Parse msgs ->" ≪ receivers[0] ≪ "\n";
    cout ≪ "mct :" ≪ m_count ≪ "\n";
    cout.flush();

}

void
TextInput::Transform_to_idealized()
{
    char dummy_text[550];
    unsigned short count;
    short k;
    Bool concat;

        cout ≪ "recers 0 before" ≪ receivers[0] ≪ "\n";
        cout.flush();
    count = 0;
    for (count=0; count < num_messages; count++)
        Fill_formulae(count);

    // Insert stars and append extensions to all formulae in messages
    Insert_stars_and_extensions();

    // Now assemble the message from the formulae
    for (count=0; count < num_messages; count++)
    {
        concat = False;
        cout ≪ "recers " ≪ receivers[count] ≪ "\n";
        cout.flush();
        sprintf(dummy_text,"fact(%hd, told(%s", count+1,
                receivers[count]);
        for(k = 0; k < idealized[count].num_formulae; k++)
        {
```

```
                strcat(dummy_text,", ");
                if ((k == 0) && (idealized[count].num_formulae > 1))
                {
                    strcat(dummy_text, "[");
                    concat = True;
                }
                strcat(dummy_text, idealized[count].formulae[k]);
            }
            if (concat)
                strcat(dummy_text, "]");
            strcat(dummy_text, "), reason([], 'Step')).");

            idealized[count].text = new char[strlen(dummy_text) + 1];
            strcpy(idealized[count].text, dummy_text);
        }
}

void
TextInput::Fill_formulae(unsigned short count)
{
    char *tempString, *str;
    char key[15];
    short i,k,j;

    tempString = messages[count];

    i = k = j = 0;

    while (*tempString)
    {
        // Does formula begin with "enc"
        if (((*tempString == 'e') && (*(tempString + 1) == 'n')
                && (*(tempString + 2) == 'c')))
        {
            str = tempString + 4;
            while (*str ≠ ')')
                key[j++] = *str++;
            str++;
            key[j] = NULL;

            tempString = str++;

            while (*tempString ≠ '{')
                tempString++;

            strcpy(idealized[count].formulae[i], "encrypt(");
            k += 8;
            tempString++;
            while (*tempString ≠ '}')
                idealized[count].formulae[i][k++] = *tempString++;
            idealized[count].formulae[i][k] = NULL;
            idealized[count].num_formulae++;

            strcat(idealized[count].formulae[i], ", ");
            strcat(idealized[count].formulae[i], key);
            strcat(idealized[count].formulae[i], "))");
            j = 0;
```

```
        k = 0;
        i++;

    }
    else
    // Does it begin with "dec"
    if ((*tempString == 'd') && (*(tempString + 1) == 'e')
            && (*(tempString + 2) == 'c'))
    {
        str = tempString + 4;
        while (*str ≠ ')')
            key[j++] = *str++;
        str++;
        key[j] = NULL;

        tempString = str++;

        while (*tempString ≠ '{')
            tempString++;

        strcpy(idealized[count].formulae[i], "decrypt(");
        k += 8;
        tempString++;
        while (*tempString ≠ '}')
            idealized[count].formulae[i][k++] = *tempString++;
        idealized[count].formulae[i][k] = NULL;
        idealized[count].num_formulae++;

        strcat(idealized[count].formulae[i], ", ");
        strcat(idealized[count].formulae[i], key);
        strcat(idealized[count].formulae[i], "))");
        k = 0;
        i++;
    }
    // Does it begin with "pub"
    if ((*tempString == 'p') && (*(tempString + 1) == 'u')
            && (*(tempString + 2) == 'b'))
    {
        str = tempString + 4;
        while (*str ≠ ')')
            key[j++] = *str++;
        str++;
        key[j] = NULL;

        tempString = str++;

        while (*tempString ≠ '{')
            tempString++;

        strcpy(idealized[count].formulae[i], "encrypt(");
        k += 8;
        tempString++;
        while (*tempString ≠ '}')
            idealized[count].formulae[i][k++] = *tempString++;
        idealized[count].formulae[i][k] = NULL;
        idealized[count].num_formulae++;
```

```
        strcat(idealized[count].formulae[i], ", ");
        strcat(idealized[count].formulae[i], key);
        strcat(idealized[count].formulae[i], "))");
        k = 0;
        i++;
    }
    // Does it start with "pri"
    if ((*tempString == 'p') && (*(tempString + 1) == 'r')
            && (*(tempString + 2) == 'i'))
    {
        str = tempString + 4;
        while (*str ≠ ')')
            key[j++] = *str++;
        str++;
        key[j] = NULL;

        tempString = str++;

        while (*tempString ≠ '{')
            tempString++;

        strcpy(idealized[count].formulae[i], "decrypt(");
        k += 8;
        tempString++;
        while (*tempString ≠ '}')
            idealized[count].formulae[i][k++] = *tempString++;
        idealized[count].formulae[i][k] = NULL;
        idealized[count].num_formulae++;

        strcat(idealized[count].formulae[i], ", ");
        strcat(idealized[count].formulae[i], key);
        strcat(idealized[count].formulae[i], "))");
        k = 0;
        i++;
    }
    else
    // Does it start with "h(", for hashed
    if ((*tempString == 'h') && (*(tempString + 1) == '{'))
    {
        tempString += 2;
        strcat(idealized[count].formulae[i], "h(");
        k += 2;

        while (*tempString ≠ '}')
            idealized[count].formulae[i][k++] = *tempString++;
        idealized[count].formulae[i][k++] = ')';
        idealized[count].formulae[i][k] = NULL;
        idealized[count].num_formulae++;
        i++;
        k = 0;
    }
    else
    // Next component ?
    if ((*tempString ≠ ',') && (!isspace(*tempString)) &&
            isalnum(*tempString))
    {
        while ((*tempString ≠ ',') && (!isspace(*tempString))
```

```
                    && (*tempString ≠ NULL))
                idealized[count].formulae[i][k++] = *tempString++;

            idealized[count].formulae[i][k] = NULL;
            idealized[count].num_formulae++;
            i++;
            k = 0;
        }

        if (*tempString ≠ NULL)
            tempString++;
    }
}


void
TextInput::Insert_stars_and_extensions()
{
    short i,j,k,a, sender_index;
    char tempstr[550];
    Bool Sent;

    for (i=0; i < num_messages; i++)
        for(a=0; a < idealized[i].num_formulae; a++)
        {
            // Valid component for insertion of stars ?
            if (!ValidFormula(idealized[i].formulae[a]))
                continue;
            Sent = False;
            for (j=0; j < i; j++)
                if (!strcmp(senders[j], receivers[i]))
                    for (k=0; k < idealized[j].num_formulae; k++)
                        if (!strcmp(idealized[j].formulae[k],
                            idealized[i].formulae[a]))
                            Sent = True;

            if (!Sent)
            {
                // If not sent before
                for (k=0; k < num_messages; k++)
                    if (!strcmp(senders[k], senders[i]))
                    {
                        sender_index = k;
                        break;
                    }
                memset(tempstr, 0, 550);

                // Build translated formula
                strcpy(tempstr, "ext(star(");
                strcat(tempstr, idealized[i].formulae[a]);
                strcat(tempstr, ")");
                if (sender_assumptions[sender_index] ≠ "")
                {
                    strcat(tempstr, ", ");
                    strcat(tempstr, sender_assumptions[sender_index]);
                    strcat(tempstr, ")");
                }
```

```
                else
                    strcat(tempstr, ", nil)");
                strcpy(idealized[i].formulae[a], tempstr);
            }
        }
}

void
TextInput::Load_assumptions()
{
    FILE *fp;
    char string[LINELENGTH];
    char *str1, *str2, *str3;
    char belief[500], sender[ENTITY_LENGTH];
    unsigned short a_index, no_beliefs[MAX_SENDERS], i, j;

    // Open initial assumptions file
    fp = fopen(fassumptionsfile, "r+");

    if (fp == NULL)
    {
        cerr << "Could not open file " << fassumptionsfile << "\n";
        exit(1);
    }

    // Initialise no_beliefs to all 0..
    for (i=0; i < MAX_SENDERS; i++)
        no_beliefs[i] = 0;

    fgets(string, LINELENGTH, fp);
    while (!feof(fp))
    {

        memset(belief, 0, 80);
        if ((str1 = strstr(string, POSSESSES)) ||
                (str1 = strstr(string, CONVEYED)) ||
                (str1 = strstr(string, BELIEVES)))
        {
            if ((str2 = strstr(str1, REASON)) == NULL)
                str2 = strstr(str1, REASON_SP);

            // Get the belief from input string
            strncpy(belief, str1, abs(str2-str1));
            str3 = strstr(belief, "(");
            str3++;
            i = 0;
            while (*str3 ≠ ',')
                sender[i++] = *str3++;
            sender[i] = NULL;

            // Get index of sender
            for (j=0; j < num_messages; j++)
                if (!strcasecmp(senders[j], sender))
                {
                    a_index = j;
                    break;
                }
```

```
            if (no_beliefs[a_index] > 0)
                strcat(sender_assumptions[a_index], ", ");

            strcat(sender_assumptions[a_index], belief);
            no_beliefs[a_index] += 1;
        }
        fgets(string, LINELENGTH, fp);
    }
}

unsigned short
TextInput::return_num_nodes() const
{
    unsigned short i, j;
    unsigned short num_senders, num_receivers;

    // Number of nodes cannot be more than this
    num_senders = num_receivers = num_messages;

    for (i=0; i < num_messages; i++)
        for (j=(i+1); j < num_messages; j++)
        {
            if (!strcmp(senders[i], senders[j]))
                num_senders -= 1;
            if (!strcmp(receivers[i], receivers[j]))
                num_receivers -= 1;
        }

    // return maximum of senders and receivers
    return (num_senders > num_receivers ? num_senders : num_receivers);
}

char *
TextInput::Return_sender_name()
{
    static unsigned short count = 0;
    short i;
    Bool tag;

    tag = False;

    while (!tag && (count < num_messages))
    {
        tag = True;
        for (i=0; i < count; i++)
            if (!strcmp(senders[i], senders[count]))
            {
                tag = False;
                count++;
                break;
            }
    }

    if ((count < num_messages) && tag)
        return senders[count++];
    else
```

```
        return (char *)NULL;
}

char *
TextInput::Return_receiver_name()
{
    static unsigned short count = 0;
    short i;
    Bool tag;

    tag = False;

    while (!tag && (count < num_messages))
    {
        tag = True;
        for (i=0; i < count; i++)
            if (!strcmp(receivers[i], receivers[count]))
                tag = False;

        if (!tag)
            count++;
    }

    if ((count < num_messages) && tag)
        return receivers[count];
    else
        return (char *)NULL;
}

void
TextInput::Dump_to_file()
{
    short k,j;
    FILE *fp;
    char ano_str[20], tstr[30], tempstr[30];
    char store_1[30], store_2[30];

    for (k=0; k < num_messages; k++)
    {
        // Build temporary message files(2) for XSB commands and
        // actual idealized steps
        sprintf(tempstr,"m_%d",k);
        sprintf(tstr,"t_m_%d",k);

        unlink(tempstr);
        unlink(tstr);

        // File containing message string
        fp = fopen(tempstr,"w+");

        for (j=0; j ≤ k; j++)
            fprintf(fp, "%s \n", idealized[j].text);

        fclose(fp);

        // File for GNY input in XSB Prolog
        fp = fopen(tstr,"w+");
```

```
            fprintf(fp, "%s \n", "[basics, control, gny_mod].");
            sprintf(ano_str,"analyze(%s).",tempstr);
            fprintf(fp, "%s \n", ano_str);
            fprintf(fp, "%s \n", "listing.");

            fclose(fp);

            // Store both the file names for use later
            strcpy(msg_files[k], tempstr);
            strcpy(t_msg_files[k], tstr);
        }

}

char *
TextInput::Return_msg_file()
{
    unsigned short store;

    if (current_message < num_messages)
    {
        store = current_message;
        increment_ctr();
        // Return messages file name
        return msg_files[store];
    }
    else
        return NULL;
}

char *
TextInput::Return_t_msg_file()
{

    if (current_message < num_messages)
        // Return commands file name
        return t_msg_files[current_message];
    else
        return NULL;
}
```

Appendix B

Protocol Examples

B.1 The Voting Protocol

1. $Q \rightarrow P_i : N_q$
2. $P_i \rightarrow Q : P_i, N_i, V_i, H(N_q, < S_i >, V_i)$
3. $Q \rightarrow P_i : R, H(N_i, < S_i >, R)$

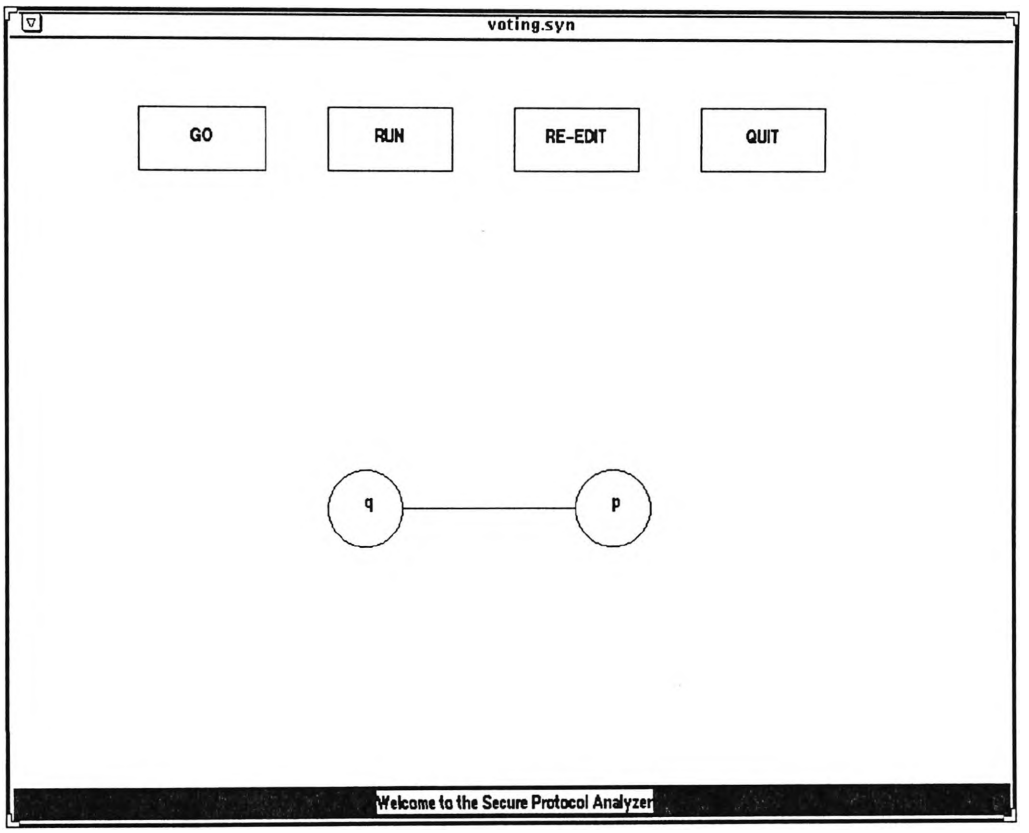
B.1.1 Protocol Representation

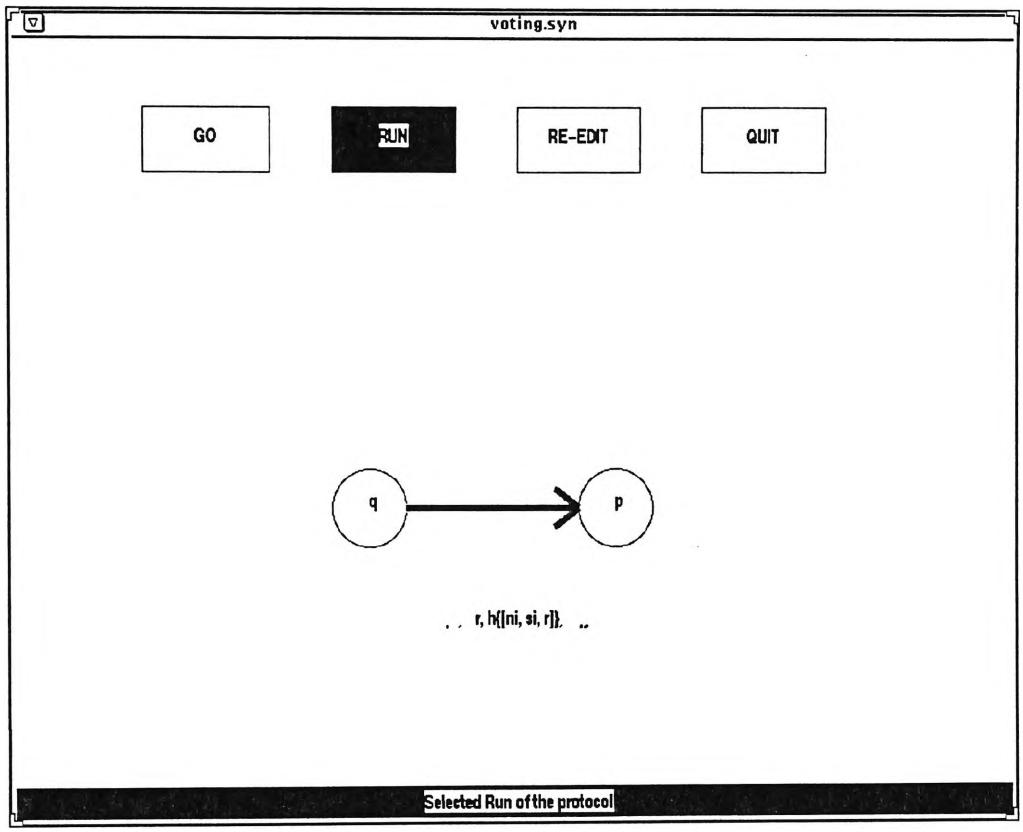
```
% Concrete Description of Voting protocol

1) q -> pi : nq
2) pi -> q : pi, ni, vi, h{[nq, si, vi]}
3) q -> pi : r, h{[ni, si, r]}

% Initial Assumptions for Voting Protocol

fact(4, possesses(pi, si), reason([], 'Assumption')).
fact(5, possesses(pi, ni), reason([], 'Assumption')).
fact(6, believes(pi, secret(q, si, pi)), reason([], 'Assumption')).
fact(8, possesses(q, si), reason([], 'Assumption')).
fact(9, possesses(q, nq), reason([], 'Assumption')).
fact(10, believes(q, secret(q, si, pi)), reason([], 'Assumption')).
fact(11, believes(q, fresh(nq)), reason([], 'Assumption')).

flag(count, 11).
```

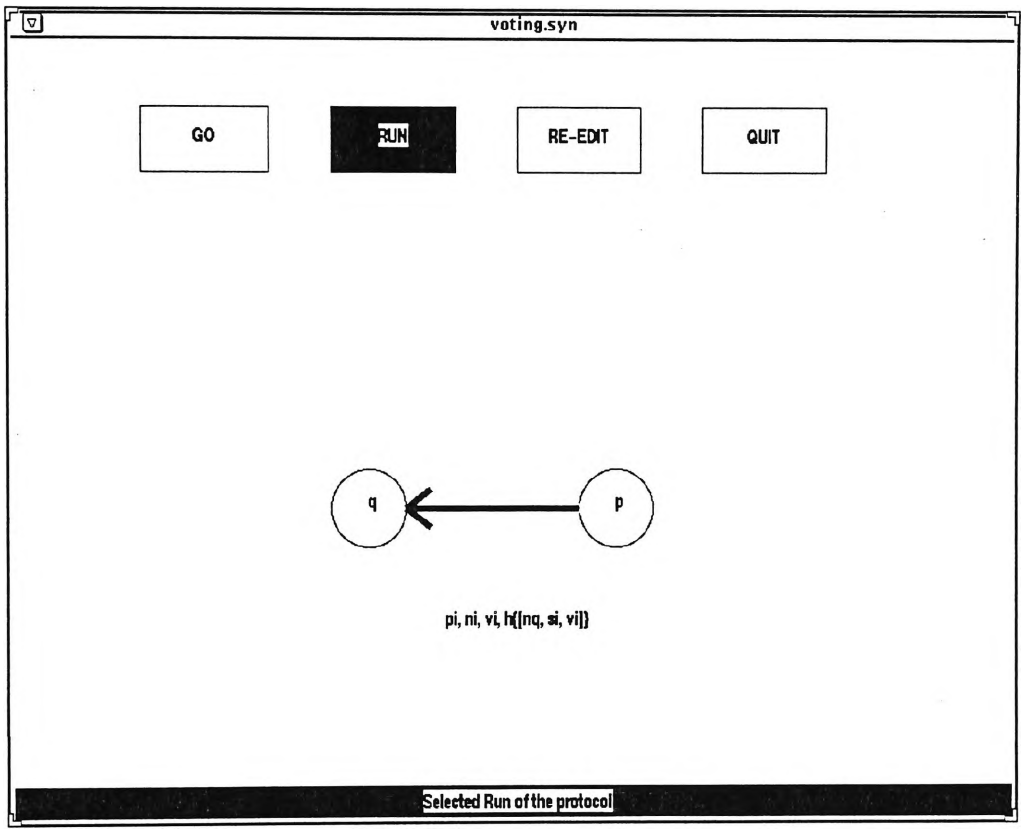
B.1.2 Output Screens

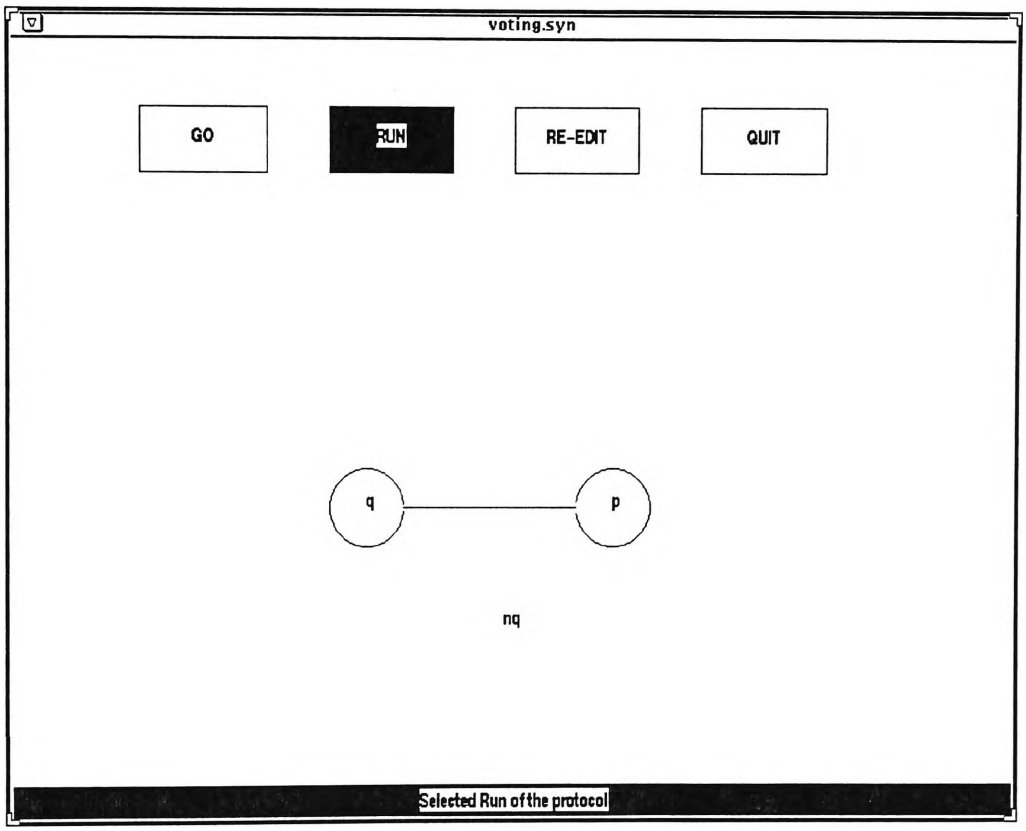
The following pages show the following states of execution of the *Protocol Analyzer* with the Voting protocol:

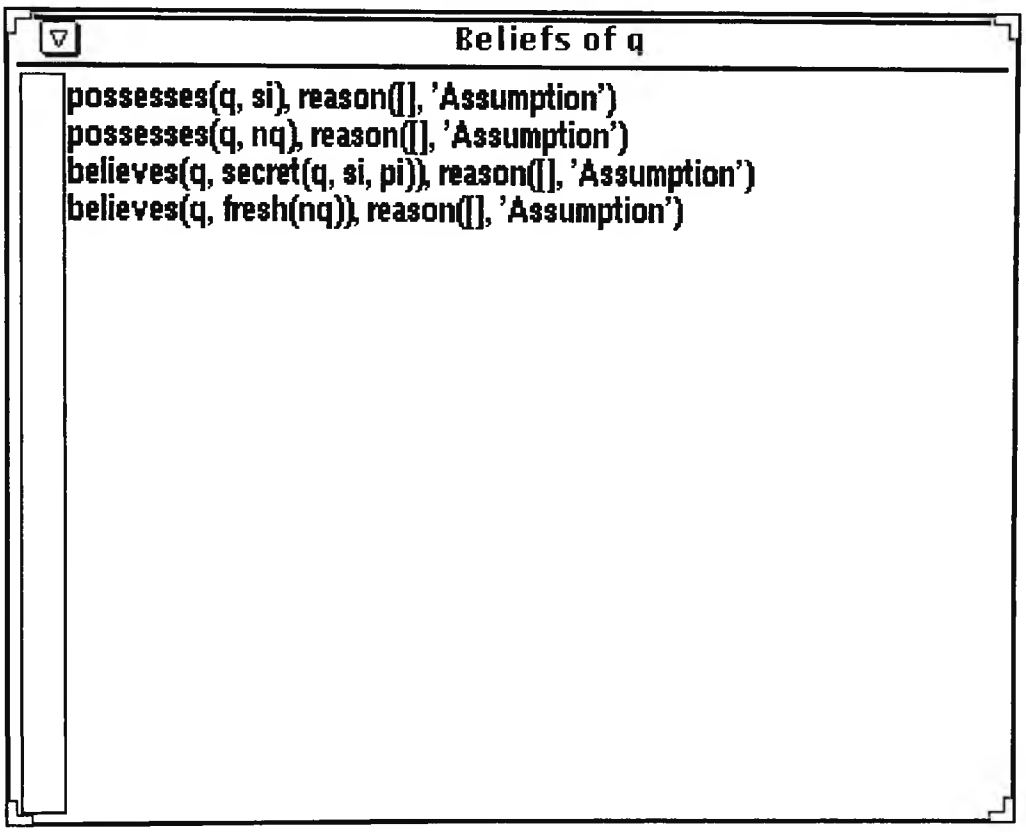
1. Main window showing protocol schematic diagram, in the initial wait state
2. Main window after step 1 of the protocol, in the *Full-Run* mode
3. Main window after step 2 of the protocol, in the *Full-Run* mode
4. Main window after step 3 of the protocol, in the *Full-Run* mode
5. Belief subwindow showing beliefs of q, in the initial wait state
6. Belief subwindow showing beliefs of pi, in the initial wait state
7. Belief subwindow showing beliefs of pi after step 1 of the protocol in *Single-Step* mode
8. Belief subwindow showing beliefs of q after step 1 of the protocol in *Single-Step* mode
9. Belief subwindow showing beliefs of pi after step 2 of the protocol in *Single-Step* mode
10. Belief subwindow showing beliefs of q after step 2 of the protocol in *Single-Step* mode
11. Belief subwindow showing beliefs of pi after step 3 of the protocol in *Single-Step* mode
12. Belief subwindow showing beliefs of q after step 3 of the protocol in *Single-Step* mode
13. Proof of a derived statement in the *Single-Step* mode
14. Window showing initial assumptions for re-editing
15. Belief subwindow showing revised beliefs of pi, after re-editing initial assumptions, in *Single-Step* mode
16. Belief subwindow showing revised beliefs of q, after re-editing initial assumptions, in *Single-Step* mode

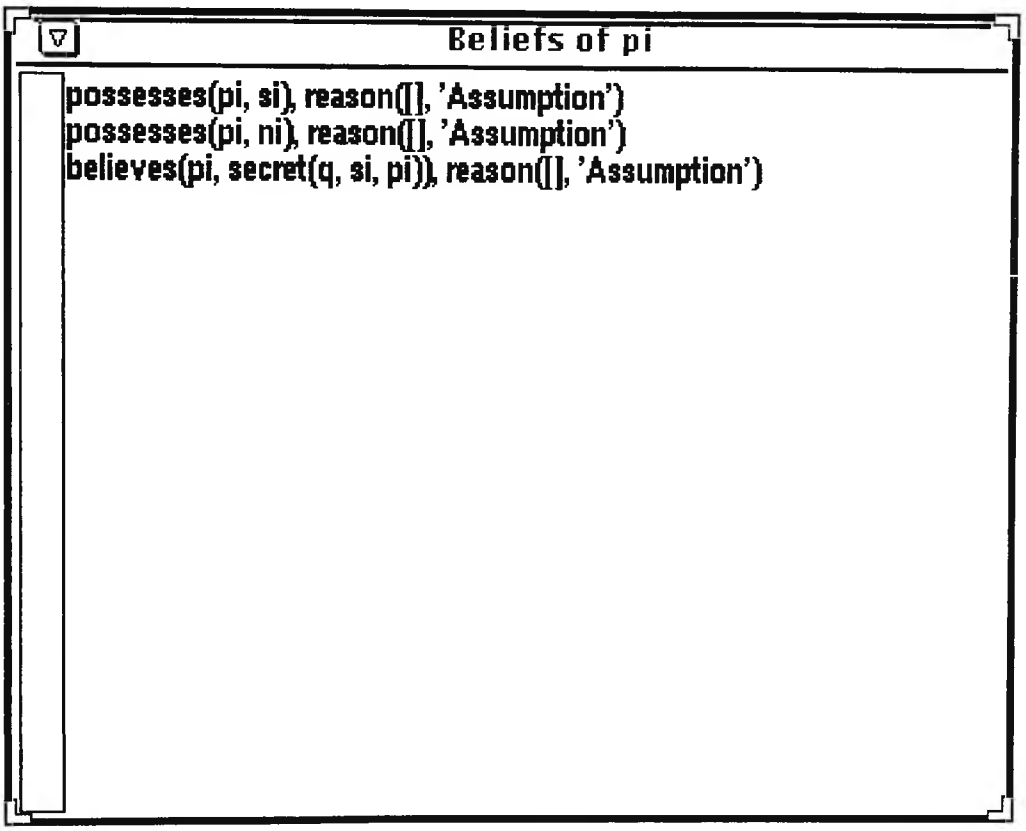

voting.syn
GO
RUN
RE-EDIT
QUIT
q
p
Welcome to the Secure Protocol Analyzer

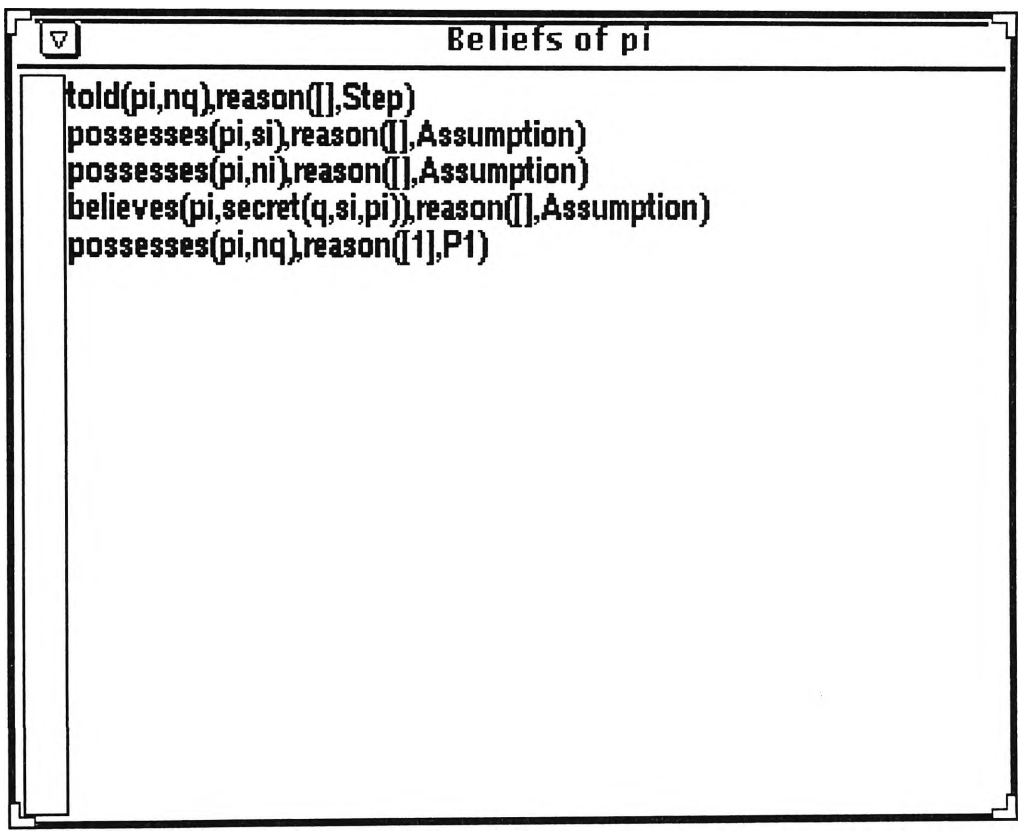

voting.syn
GO
RUN
RE-EDIT
QUIT
q
p
. . r, h([ni, si, r]). ..
Selected Run of the protocol

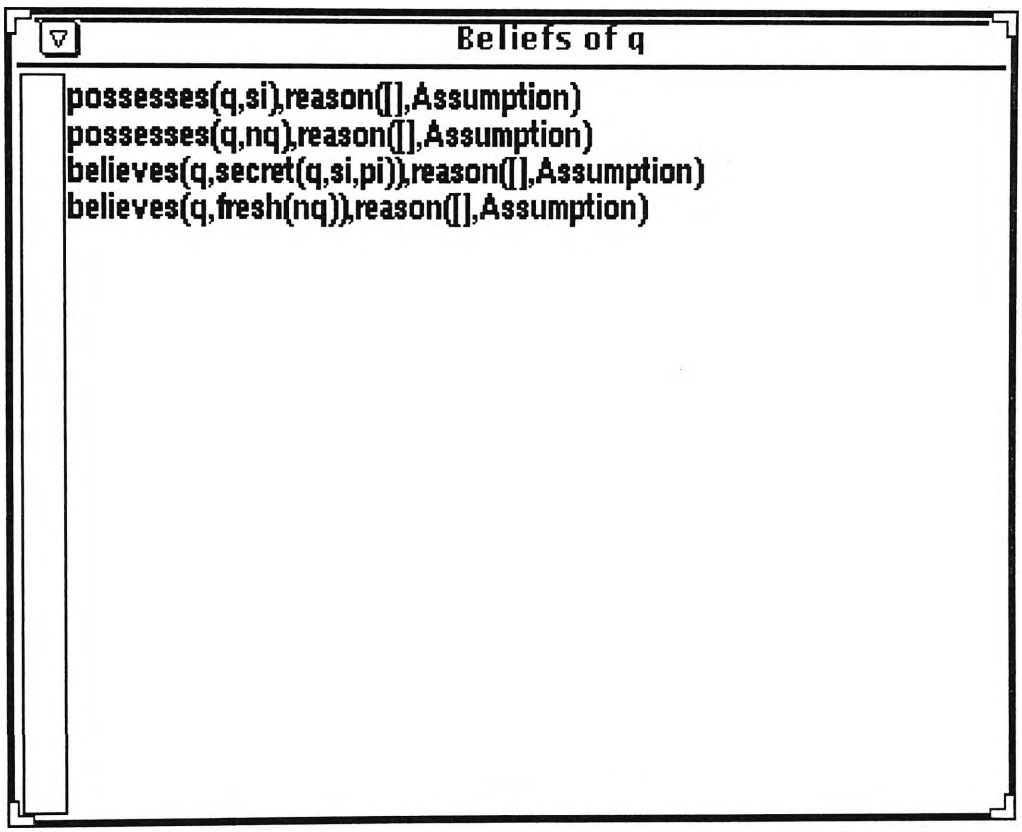

voting.syn
GO
RUN
RE-EDIT
QUIT
q
p
pi, ni, vi, h([nq, si, vi])
Selected Run of the protocol

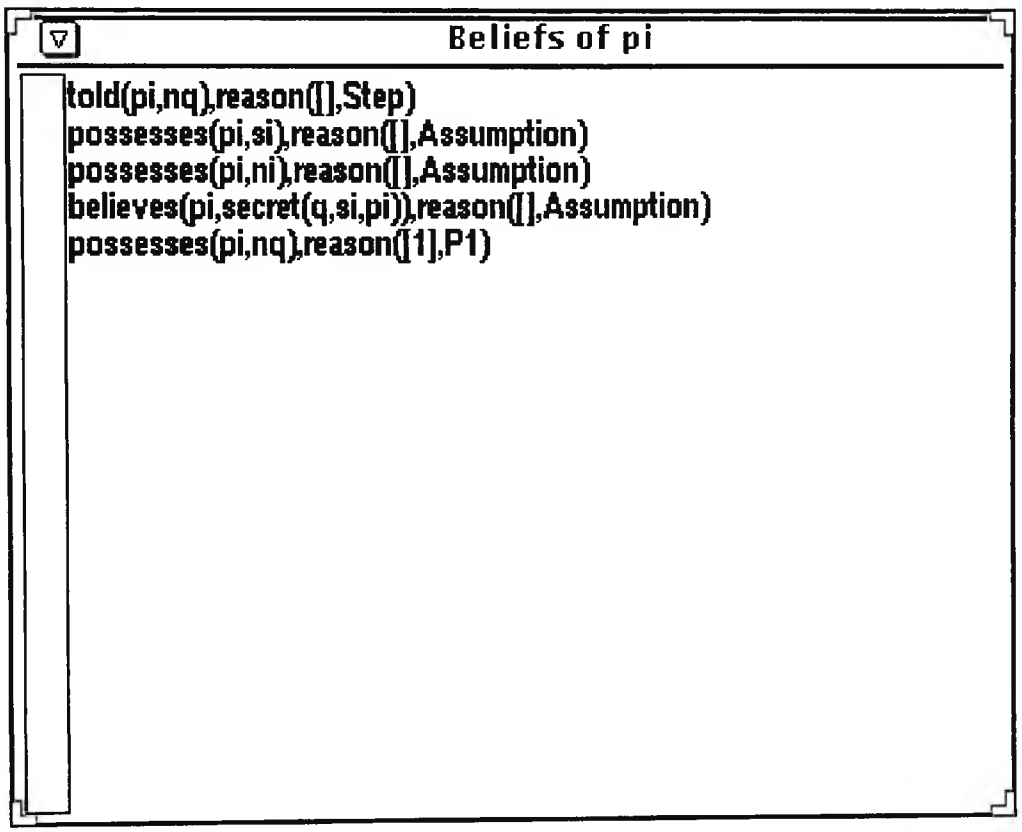

voting.syn
GO
RUN
RE-EDIT
QUIT
q
p
nq
Selected Run of the protocol

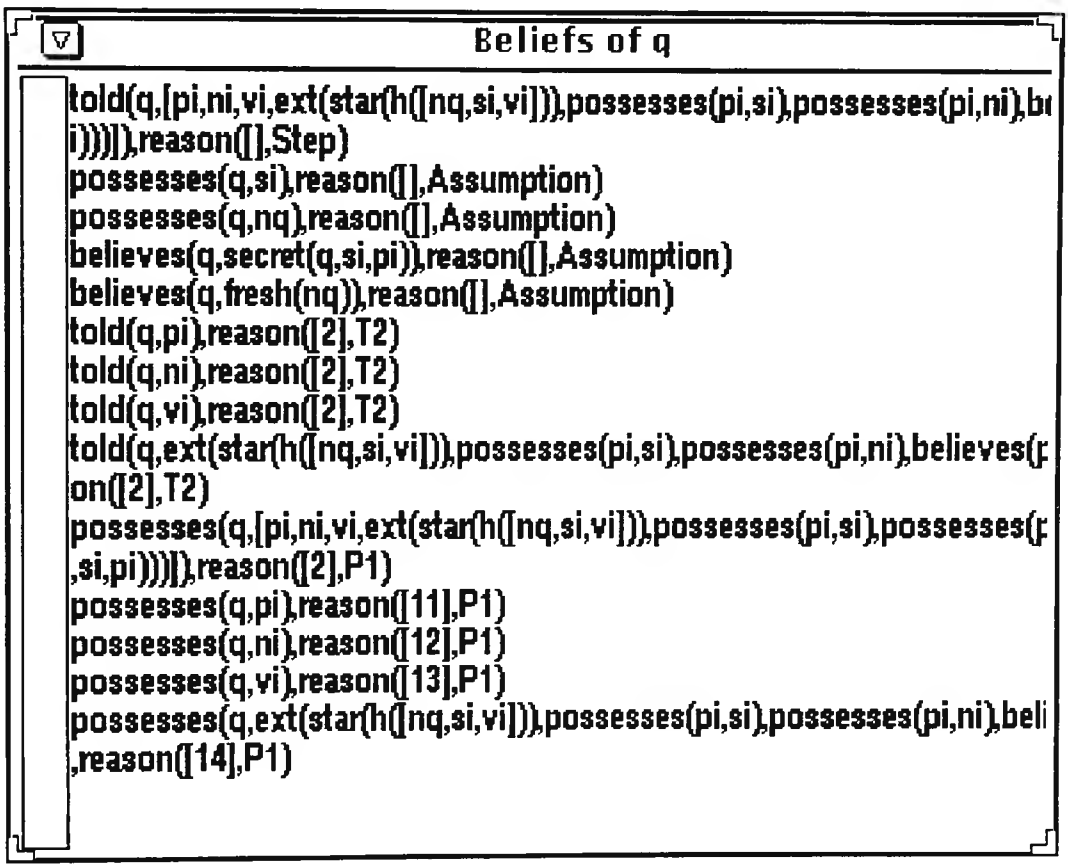

Beliefs of q
possesses(q, si), reason([], 'Assumption')
possesses(q, nq), reason([], 'Assumption')
believes(q, secret(q, si, pi)), reason([], 'Assumption')
believes(q, fresh(nq)), reason([], 'Assumption')

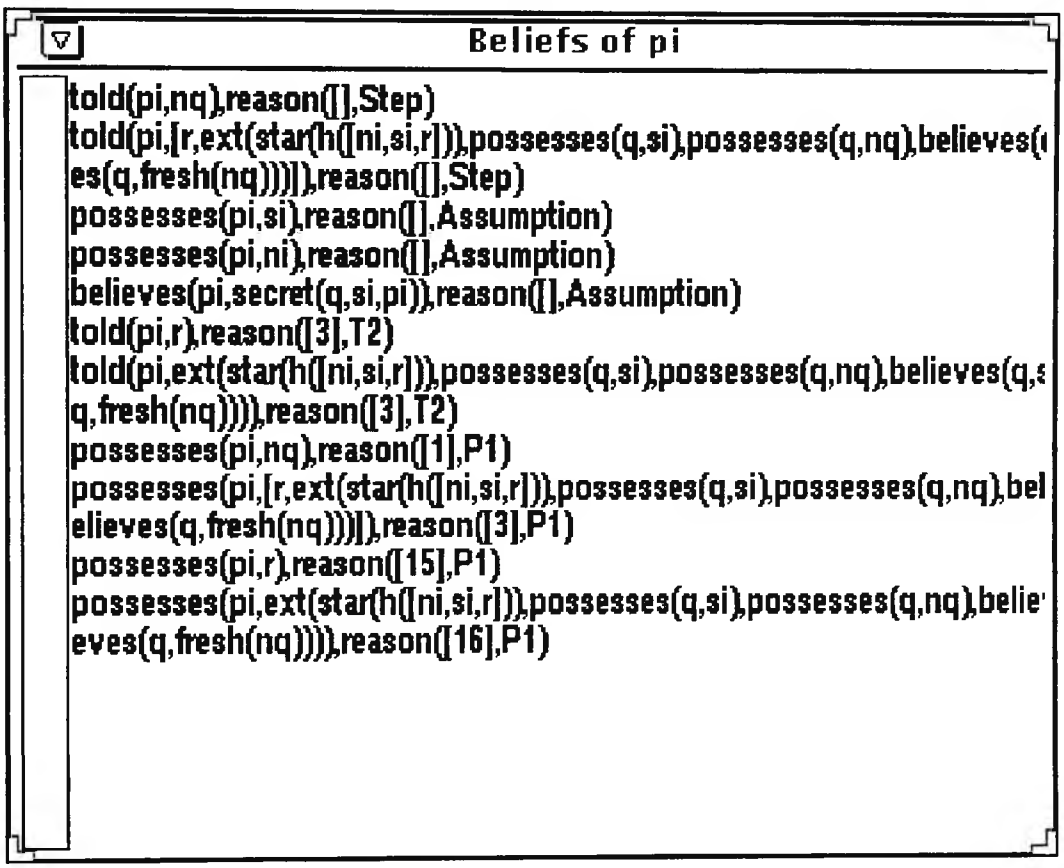

Beliefs of pi
possesses(pi, si), reason([], 'Assumption')
possesses(pi, ni), reason([], 'Assumption')
believes(pi, secret(q, si, pi)), reason([], 'Assumption')

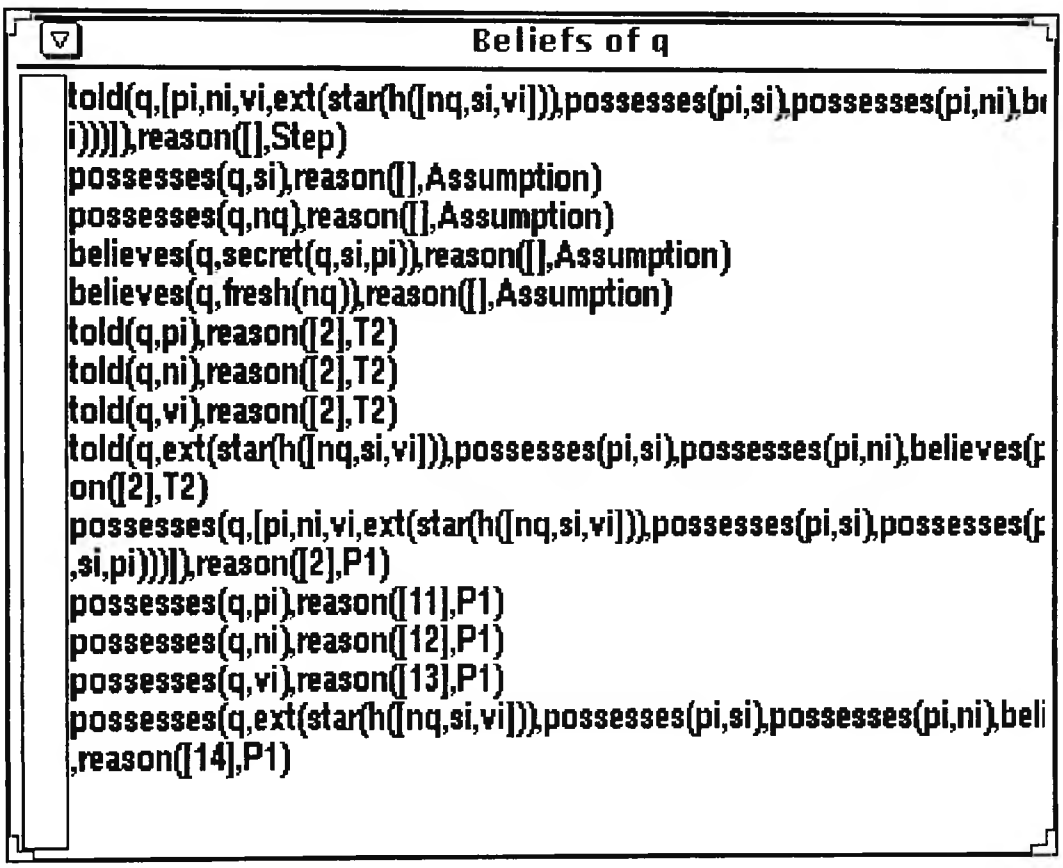

Beliefs of pi
told(pi,nq),reason([],Step)
possesses(pi,si),reason([],Assumption)
possesses(pi,ni),reason([],Assumption)
believes(pi,secret(q,si,pi)),reason([],Assumption)
possesses(pi,nq),reason([1],P1)

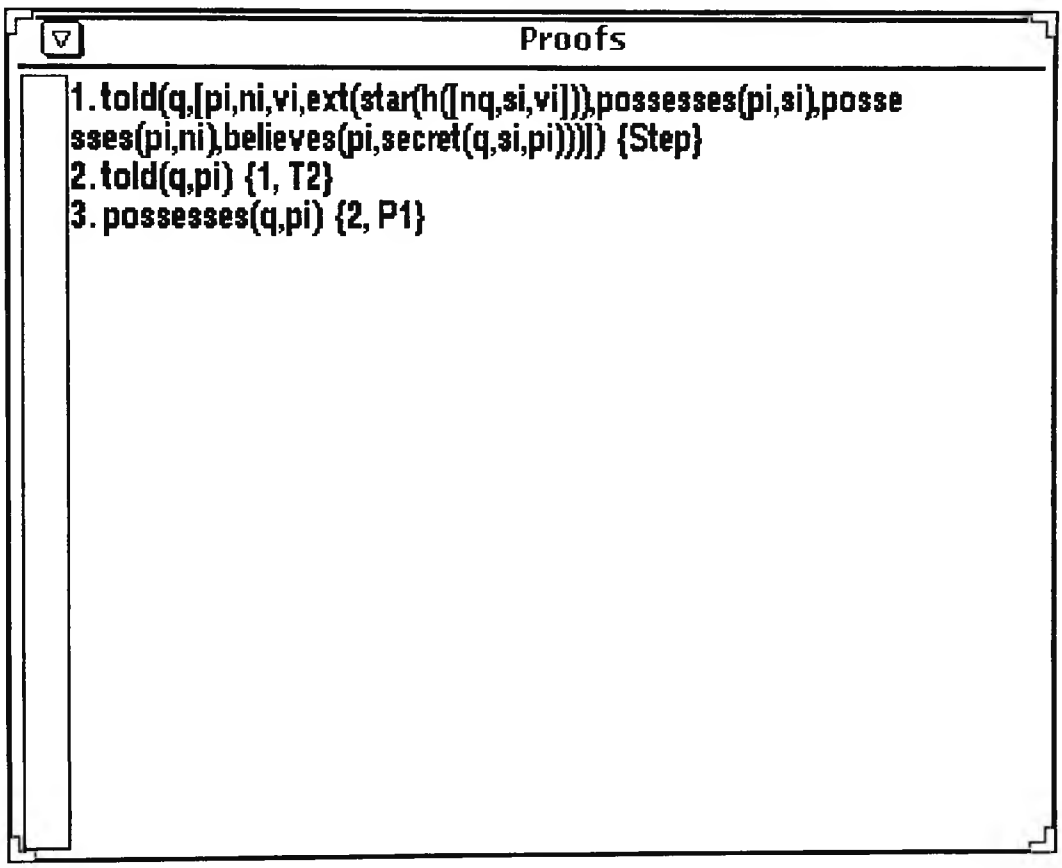

Beliefs of q
possesses(q,si),reason([],Assumption)
possesses(q,nq),reason([],Assumption)
believes(q,secret(q,si,pi)),reason([],Assumption)
believes(q,fresh(nq)),reason([],Assumption)

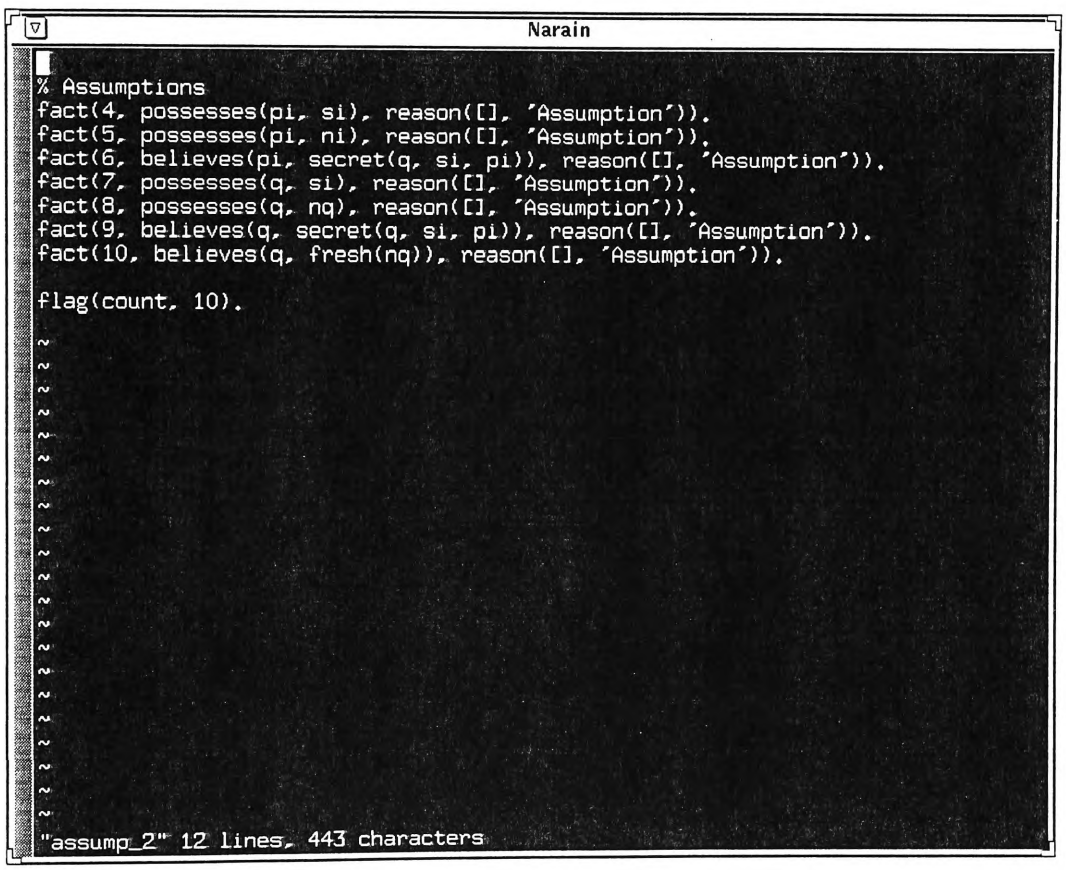

Beliefs of pi
told(pi,nq),reason([],Step)
possesses(pi,si),reason([],Assumption)
possesses(pi,ni),reason([],Assumption)
believes(pi,secret(q,si,pi)),reason([],Assumption)
possesses(pi,nq),reason([1],P1)

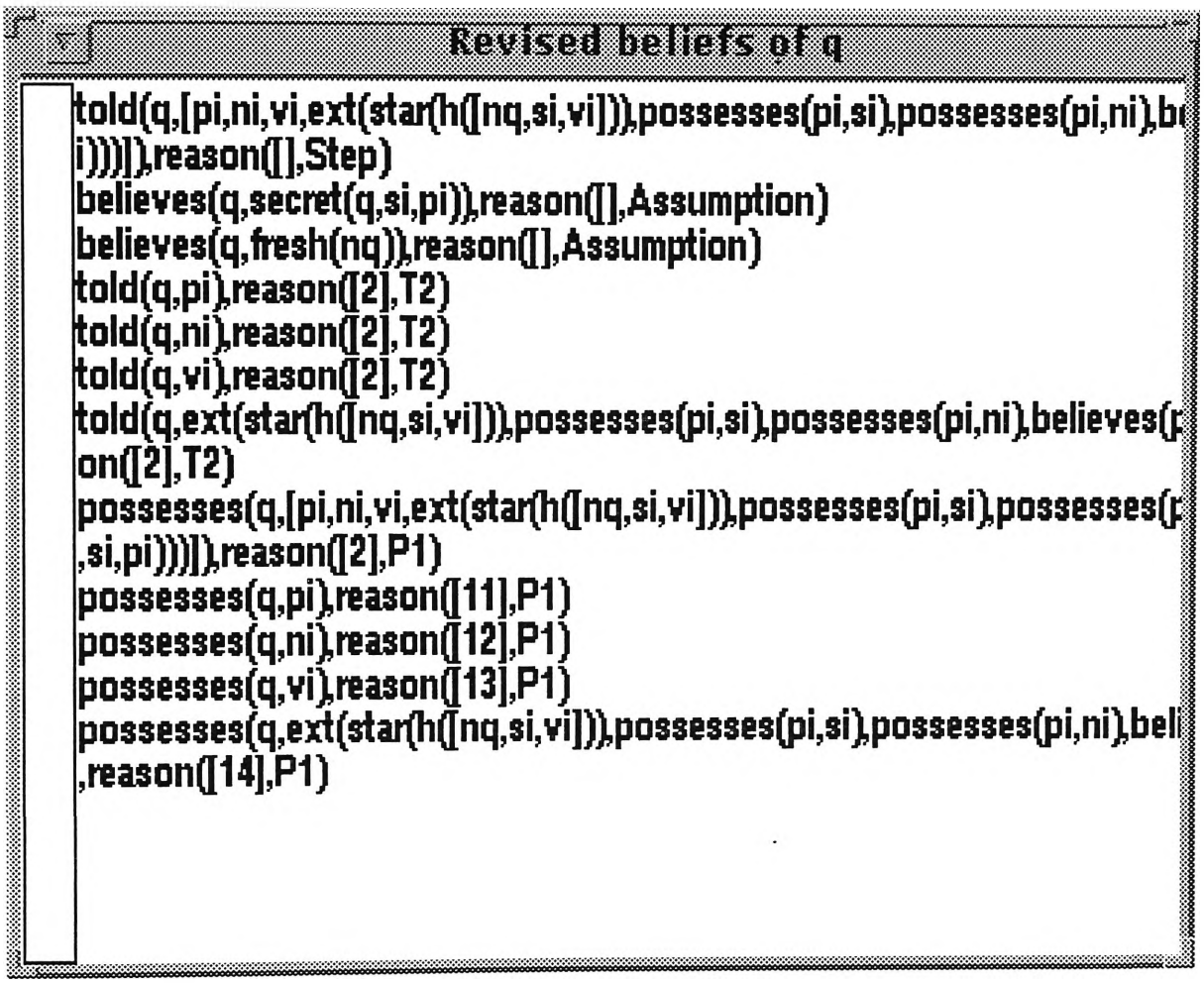

Beliefs of q
told(q,[pi,ni,vi,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),b
i))])],reason([],Step)
possesses(q,si),reason([],Assumption)
possesses(q,nq),reason([],Assumption)
believes(q,secret(q,si,pi)),reason([],Assumption)
believes(q,fresh(nq)),reason([],Assumption)
told(q,pi),reason([2],T2)
told(q,ni),reason([2],T2)
told(q,vi),reason([2],T2)
told(q,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),believes(p
on([2],T2)
possesses(q,[pi,ni,vi,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(p
,si,pi))])],reason([2],P1)
possesses(q,pi),reason([11],P1)
possesses(q,ni),reason([12],P1)
possesses(q,vi),reason([13],P1)
possesses(q,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),beli
,reason([14],P1)

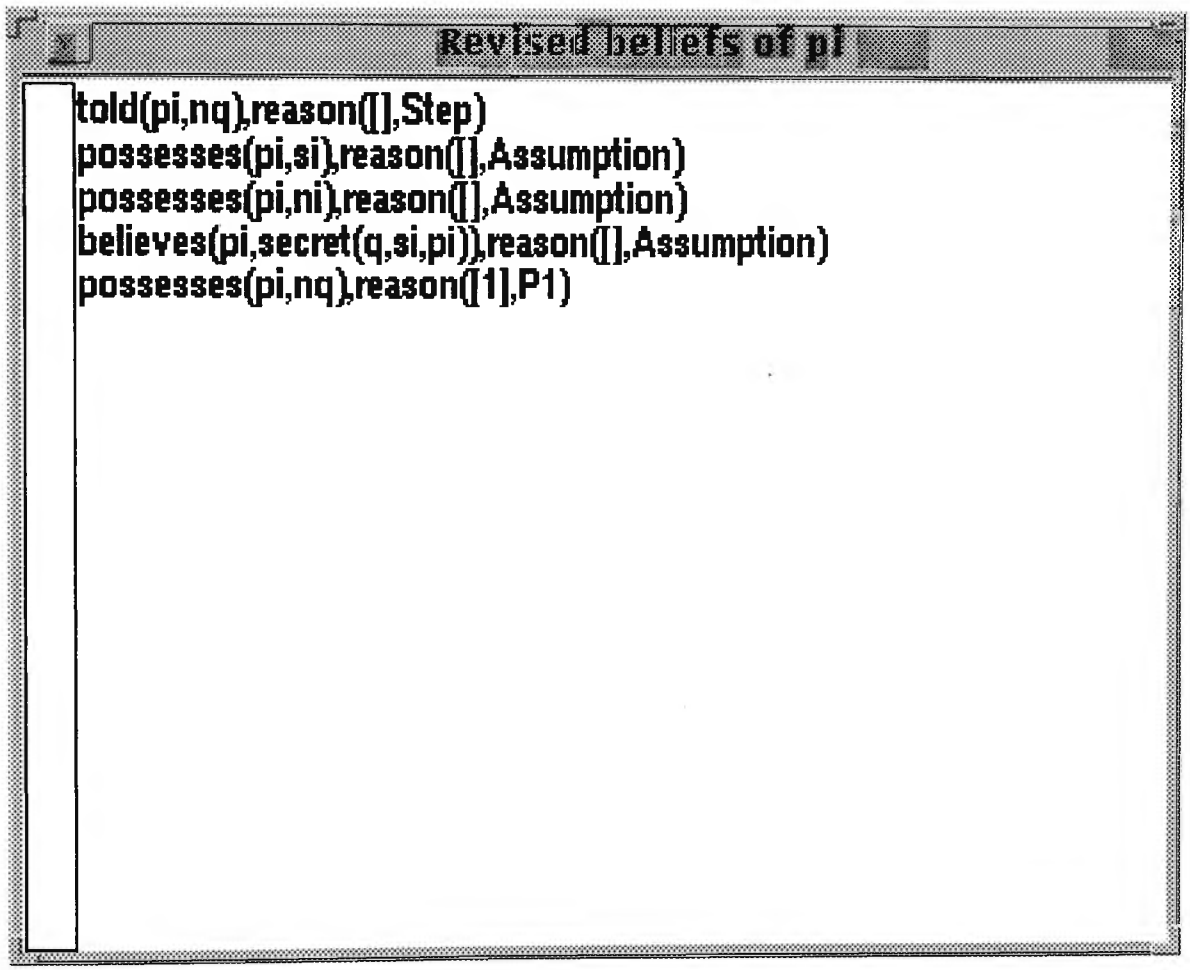

Beliefs of pi
told(pi,nq),reason([],Step)
told(pi,[r,ext(star(h([ni,si,r])),possesses(q,si),possesses(q,nq),believes(
es(q,fresh(nq))))]),reason([],Step)
possesses(pi,si),reason([],Assumption)
possesses(pi,ni),reason([],Assumption)
believes(pi,secret(q,si,pi)),reason([],Assumption)
told(pi,r),reason([3],T2)
told(pi,ext(star(h([ni,si,r])),possesses(q,si),possesses(q,nq),believes(q,
q,fresh(nq)))),reason([3],T2)
possesses(pi,nq),reason([1],P1)
possesses(pi,[r,ext(star(h([ni,si,r])),possesses(q,si),possesses(q,nq),bel
elieves(q,fresh(nq))))]),reason([3],P1)
possesses(pi,r),reason([15],P1)
possesses(pi,ext(star(h([ni,si,r])),possesses(q,si),possesses(q,nq),belie
eves(q,fresh(nq)))),reason([16],P1)


Beliefs of q
told(q,[pi,ni,vi,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),bi
i))])],reason([],Step)
possesses(q,si),reason([],Assumption)
possesses(q,nq),reason([],Assumption)
believes(q,secret(q,si,pi)),reason([],Assumption)
believes(q,fresh(nq)),reason([],Assumption)
told(q,pi),reason([2],T2)
told(q,ni),reason([2],T2)
told(q,vi),reason([2],T2)
told(q,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),believes(p
on([2],T2)
possesses(q,[pi,ni,vi,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(p
,si,pi))])],reason([2],P1)
possesses(q,pi),reason([11],P1)
possesses(q,ni),reason([12],P1)
possesses(q,vi),reason([13],P1)
possesses(q,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),beli
,reason([14],P1)


Proofs
1. told(q,[pi,ni,vi,ext(star(h([nq,si,vi])),possesses(pi,si),posse
sses(pi,ni),believes(pi,secret(q,si,pi)))]) {Step}
2. told(q,pi) {1, T2}
3. possesses(q,pi) {2, P1}


Narain
% Assumptions
fact(4, possesses(pi, si), reason([], 'Assumption')).
fact(5, possesses(pi, ni), reason([], 'Assumption')).
fact(6, believes(pi, secret(q, si, pi)), reason([], 'Assumption')).
fact(7, possesses(q, si), reason([], 'Assumption')).
fact(8, possesses(q, nq), reason([], 'Assumption')).
fact(9, believes(q, secret(q, si, pi)), reason([], 'Assumption')).
fact(10, believes(q, fresh(nq)), reason([], 'Assumption')).
flag(count, 10).
"assump_2" 12 lines, 443 characters


Revised beliefs of q
told(q,[pi,ni,vi,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),bi
i))))]),reason([],Step)
believes(q,secret(q,si,pi)),reason([],Assumption)
believes(q,fresh(nq)),reason([],Assumption)
told(q,pi),reason([2],T2)
told(q,ni),reason([2],T2)
told(q,vi),reason([2],T2)
told(q,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),believes(p
on([2],T2)
possesses(q,[pi,ni,vi,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(p
,si,pi))))]),reason([2],P1)
possesses(q,pi),reason([11],P1)
possesses(q,ni),reason([12],P1)
possesses(q,vi),reason([13],P1)
possesses(q,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),beli
,reason([14],P1)


Revised beliefs of pi
told(pi,nq),reason([],Step)
possesses(pi,si),reason([],Assumption)
possesses(pi,ni),reason([],Assumption)
believes(pi,secret(q,si,pi)),reason([],Assumption)
possesses(pi,nq),reason([1],P1)

B.2 Yahalom Protocol

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : B, \{A, N_a, N_b\}_{K_{bs}}$
3. $S \rightarrow A : \{B, K_{ab}, N_a, N_b\}_{K_{as}}, \{A, K_{ab}\}_{K_{bs}}$
4. $A \rightarrow B : \{A, K_{ab}\}_{K_{bs}}, \{N_b\}_{K_{ab}}$

B.2.1 Protocol Representation

```
% Concrete description of Yahalom protocol

1) a -> b : a, na
2) b -> s : b, enc(shared(kbs)){[a, na, nb]}
3) s -> a : enc(shared(kas)){[b, shared(kab), na, nb]},
enc(shared(kbs)){[a, shared(kab)]}
4) a -> b : enc(shared(kbs)){[a, shared(kab)]}, enc(shared(kab)){nb}

% Initial Assumptions for Yahalom Protocol

fact(5, possesses(a, shared(kas)), reason([], 'Assumption')).
fact(6, believes(a, secret(a, kas, s)), reason([], 'Assumption')).
fact(7, possesses(a, na), reason([], 'Assumption')).
fact(8, believes(a, fresh(na)), reason([], 'Assumption')).
fact(9, believes(a, recognizes(b)), reason([], 'Assumption')).
fact(10, believes(a, honest(s)), reason([], 'Assumption')).
fact(11, believes(a, controls(s, secret(a, kab, b))), reason([],
'Assumption')).

fact(12, possesses(b, shared(kbs)), reason([], 'Assumption')).
fact(13, believes(b, secret(b, kbs, s)), reason([], 'Assumption')).
fact(14, possesses(b, nb), reason([], 'Assumption')).
fact(15, believes(b, fresh(nb)), reason([], 'Assumption')).
fact(16, believes(b, recognizes(nb)), reason([], 'Assumption')).
fact(17, believes(b, recognizes(a)), reason([], 'Assumption')).
fact(18, believes(b, honest(s)), reason([], 'Assumption')).
fact(19, believes(b, honest(a)), reason([], 'Assumption')).
fact(20, believes(b, controls(s, secret(a, kab, b))), reason([],
'Assumption')).
fact(21, believes(b, controls(a, believes(s, secret(a, kab, b)))),
reason([], 'Assumption')).
fact(22, believes(b, secret(a, nb, b)), reason([], 'Assumption')).

fact(23, possesses(s, shared(kas)), reason([], 'Assumption')).
fact(24, believes(s, secret(a, kas, s)), reason([], 'Assumption')).
fact(25, possesses(s, shared(kbs)), reason([], 'Assumption')).
fact(26, believes(s, secret(b, kbs, s)), reason([], 'Assumption')).
```

```
fact(27, possesses(s, shared(kab)), reason([], 'Assumption')).
fact(28, believes(s, secret(a, kab, b)), reason([], 'Assumption')).

flag(count, 28).
```

B.2.2 Output Screens

The following pages show the following stages in the execution of the *Protocol Analyzer* with the Yahalom protocol:

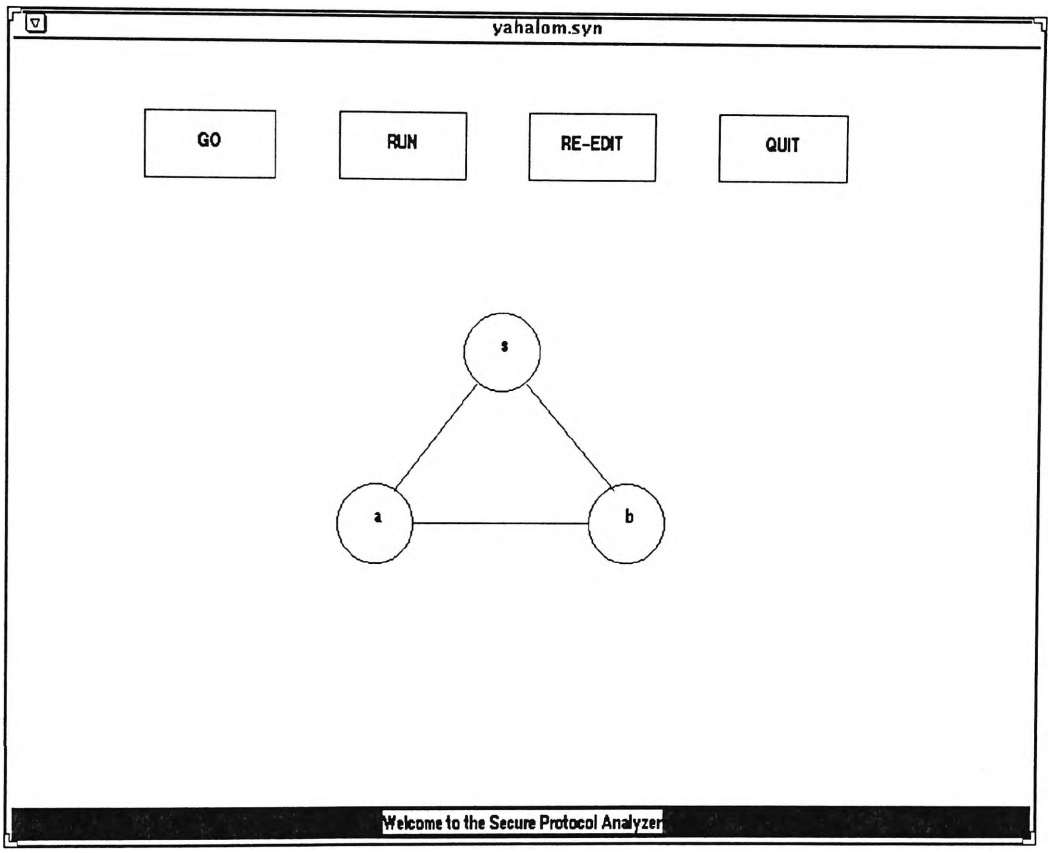
1. Main window showing the protocol schematic diagram, in the initial wait state
2. Main window after step 1 of the protocol, in the *Full-Run* mode
3. Main window after step 2 of the protocol, in the *Full-Run* mode
4. Main window after step 3 of the protocol, in the *Full-Run* mode
5. Main window after step 4 of the protocol, in the *Full-Run* mode
6. Belief subwindow showing the beliefs of a, in the initial wait state
7. Belief subwindow showing the beliefs of b, in the initial wait state
8. Belief subwindow showing the beliefs of s, in the initial wait state
9. Belief subwindow showing the beliefs of a after step 1 of the protocol, in the *Single-Step* mode
10. Belief subwindow showing the beliefs of b after step 1 of the protocol, in the *Single-Step* mode
11. Belief subwindow showing the beliefs of s after step 1 of the protocol, in the *Single-Step* mode
12. Belief subwindow showing the beliefs of a after step 2 of the protocol, in the *Single-Step* mode
13. Belief subwindow showing the beliefs of b after step 2 of the protocol, in the *Single-Step* mode
14. Belief subwindow showing the beliefs of s after step 2 of the protocol, in the *Single-Step* mode
15. Belief subwindow showing the beliefs of a after step 3 of the protocol, in the *Single-Step* mode
16. Belief subwindow showing the beliefs of b after step 3 of the protocol, in the *Single-Step* mode
17. Belief subwindow showing the beliefs of s after step 3 of the protocol, in the *Single-Step* mode

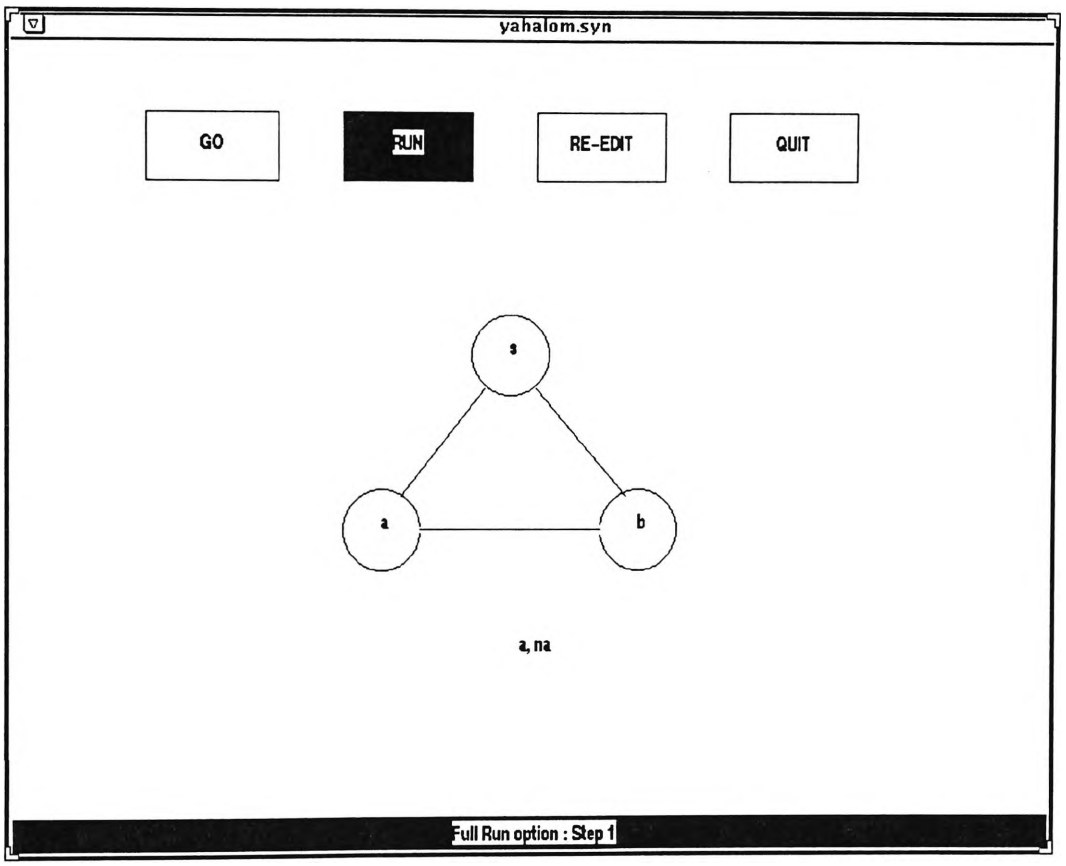
18. Belief subwindow showing the beliefs of a after step 4 of the protocol, in the *Single-Step* mode

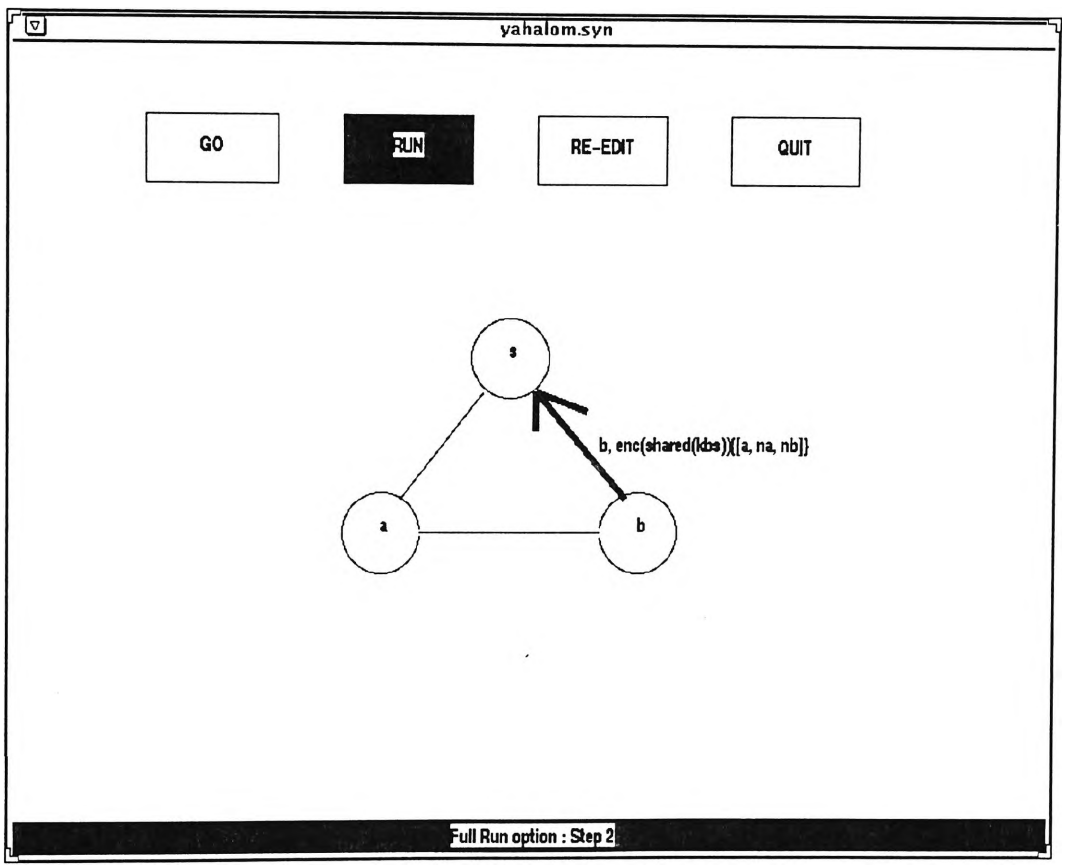
19. Belief subwindow showing the beliefs of b after step 4 of the protocol, in the *Single-Step* mode

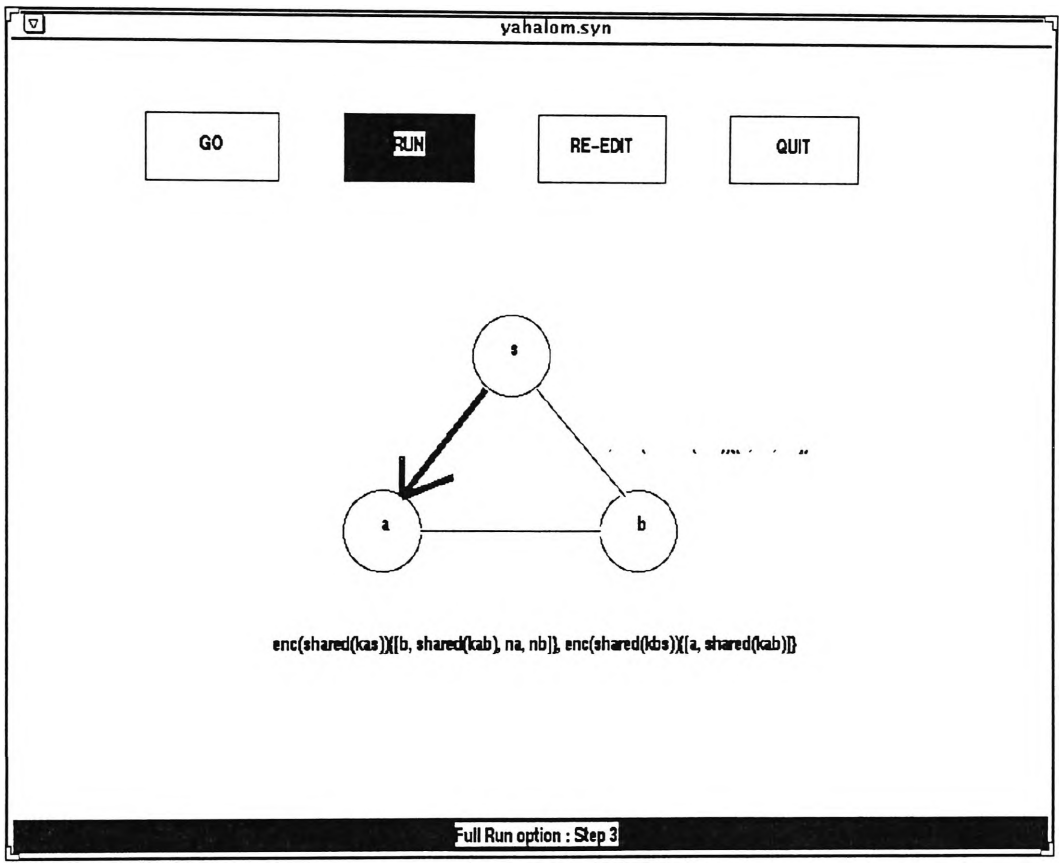
20. Belief subwindow showing the beliefs of s after step 4 of the protocol, in the *Single-Step* mode

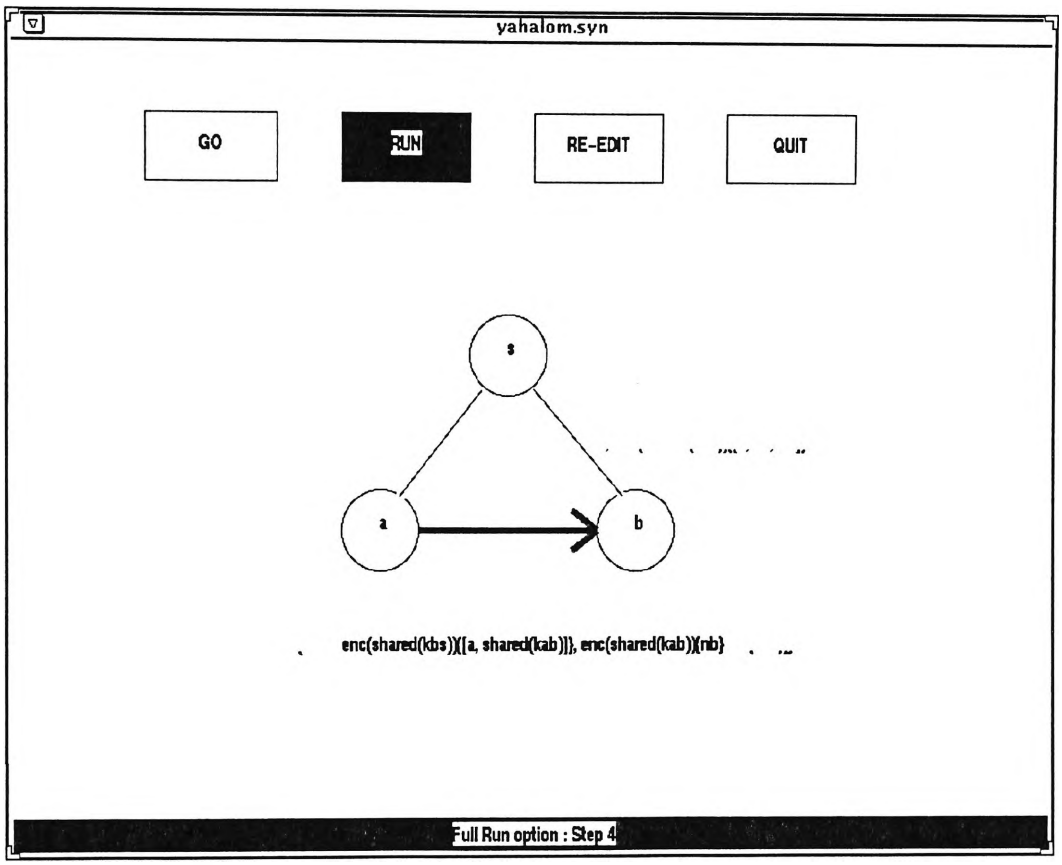
21. Proof of a derived statement in the *Single-Step* mode

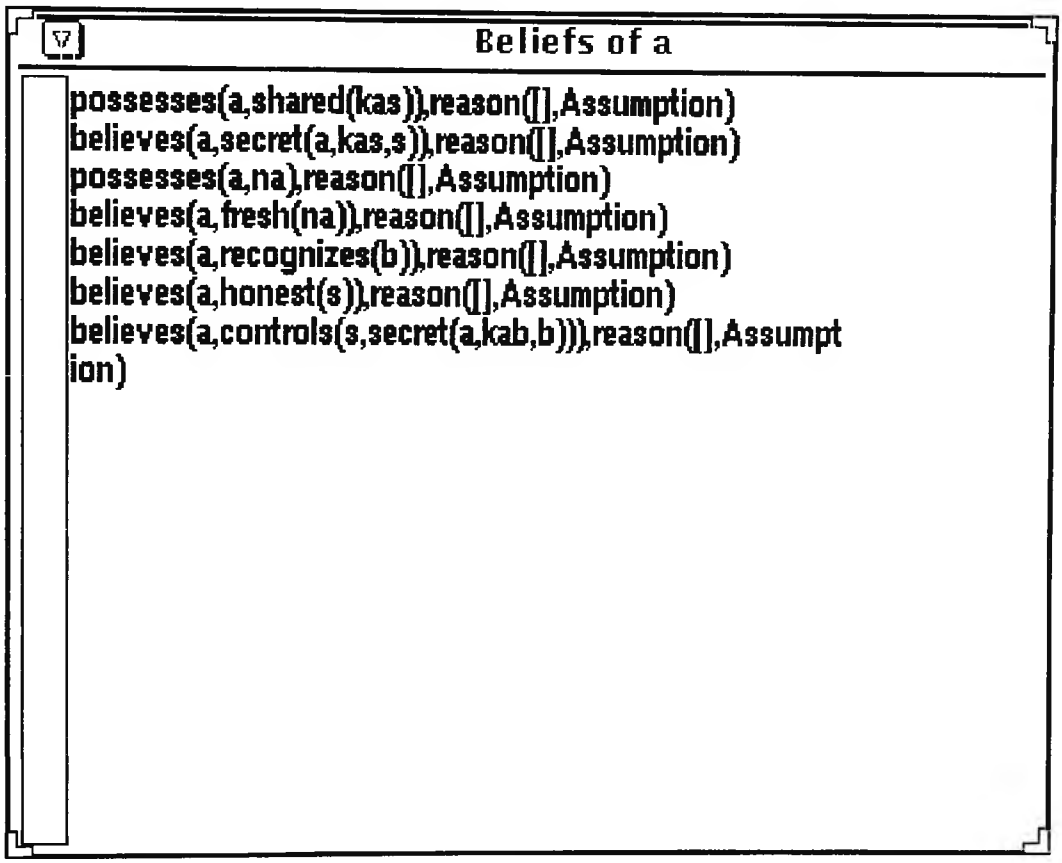

yahalom.syn
GO
RUN
RE-EDIT
QUIT
s
a
b
Welcome to the Secure Protocol Analyzer

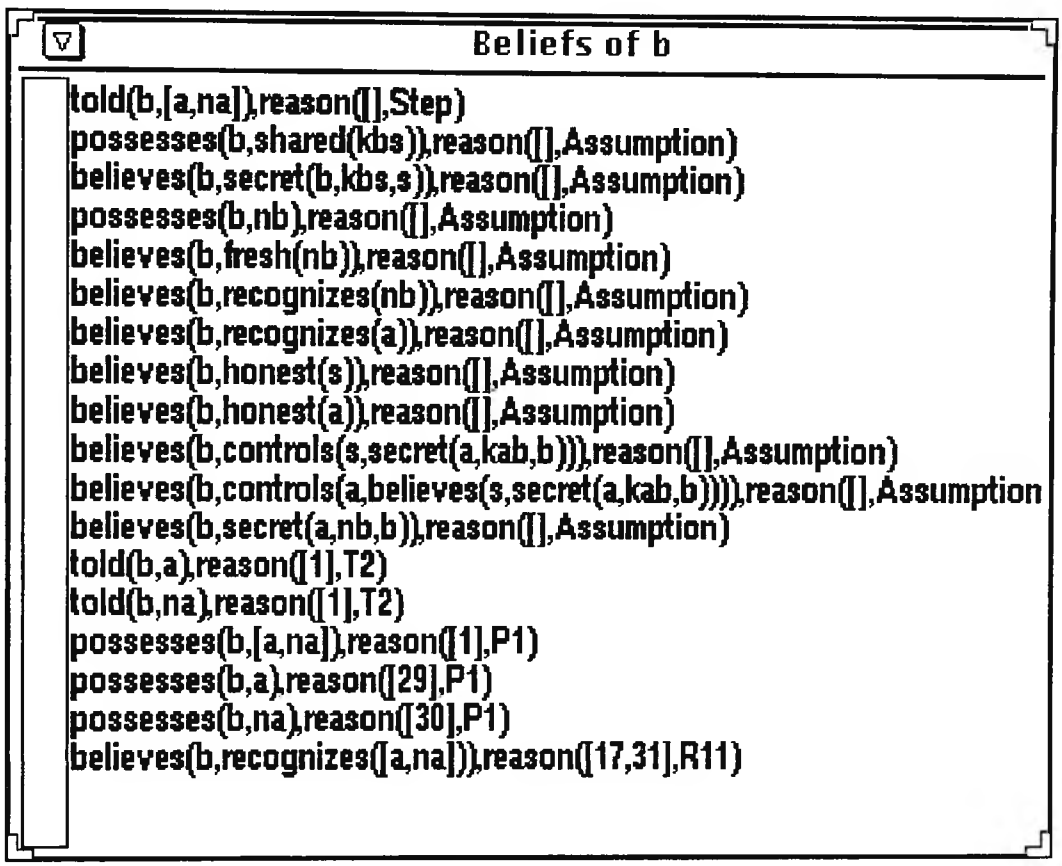

yahalom.syn
GO
RUN
RE-EDIT
QUIT
s
a
b
a, na
Full Run option : Step 1

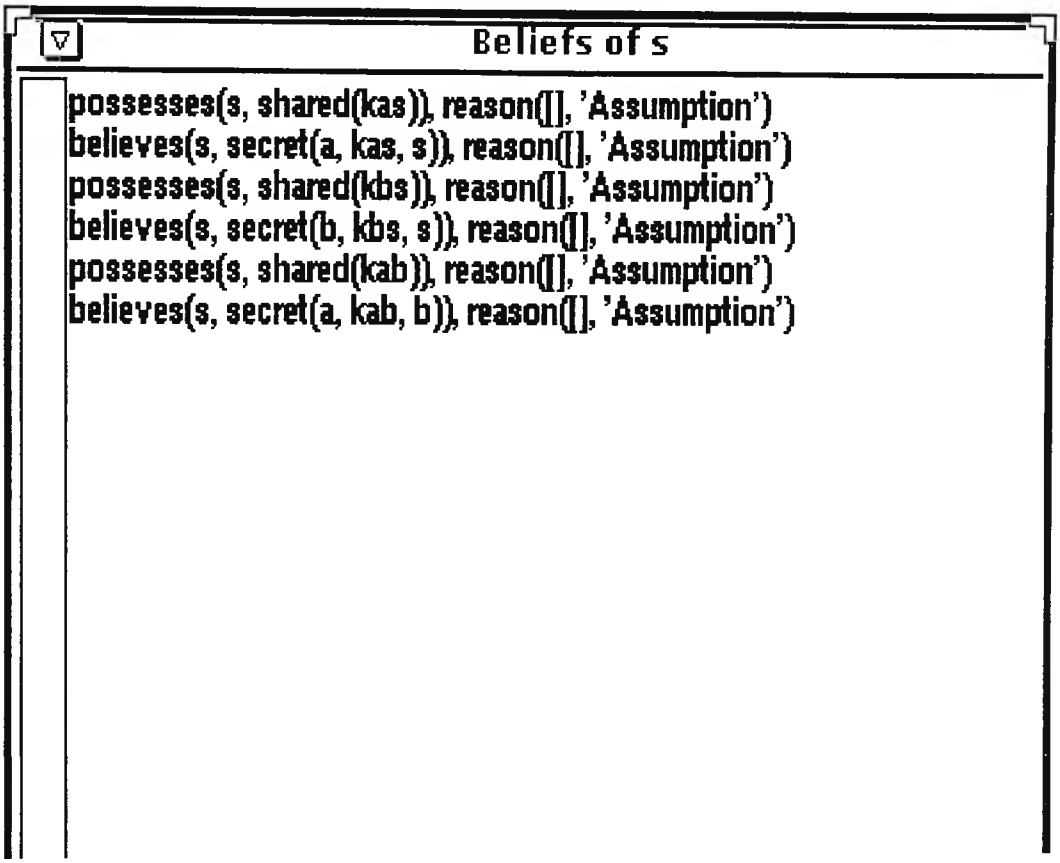

yahalom.syn
GO
RUN
RE-EDIT
QUIT
s
a
b
b, enc(shared(kbs))([a, na, nb])
Full Run option : Step 2

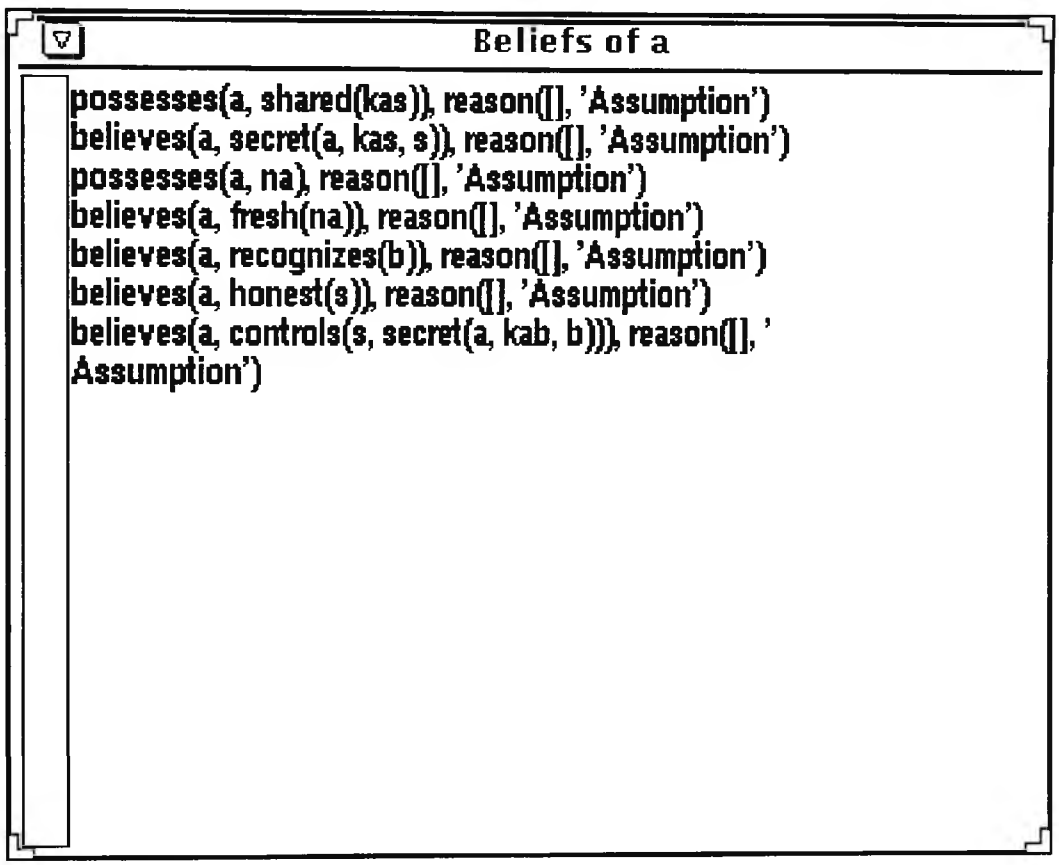

yahalom.syn
GO
RUN
RE-EDIT
QUIT
s
a
b
enc(shared(kas)){[b, shared(kab), na, nb]}, enc(shared(kbs)){[a, shared(kab)]}
Full Run option : Step 3

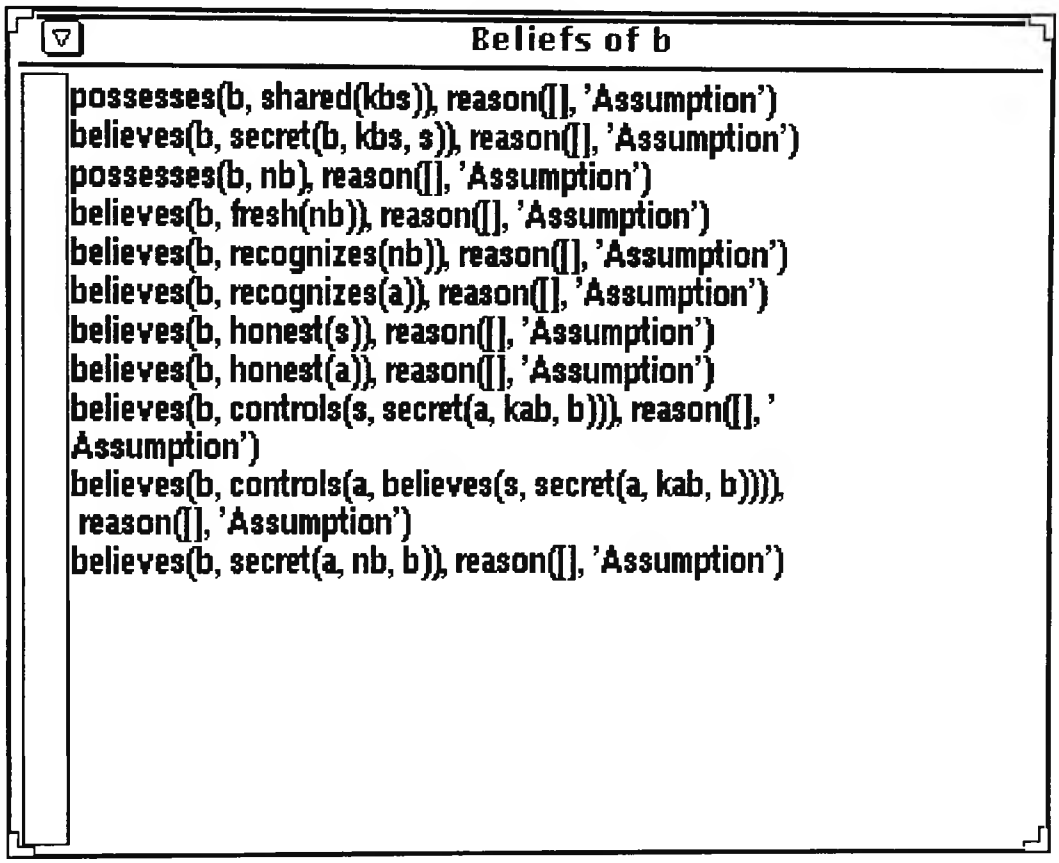

yahalom.syn
GO
RUN
RE-EDIT
QUIT
s
a
b
enc(shared(kbs)|[a, shared(kab)]), enc(shared(kab)|{nb}
Full Run option : Step 4

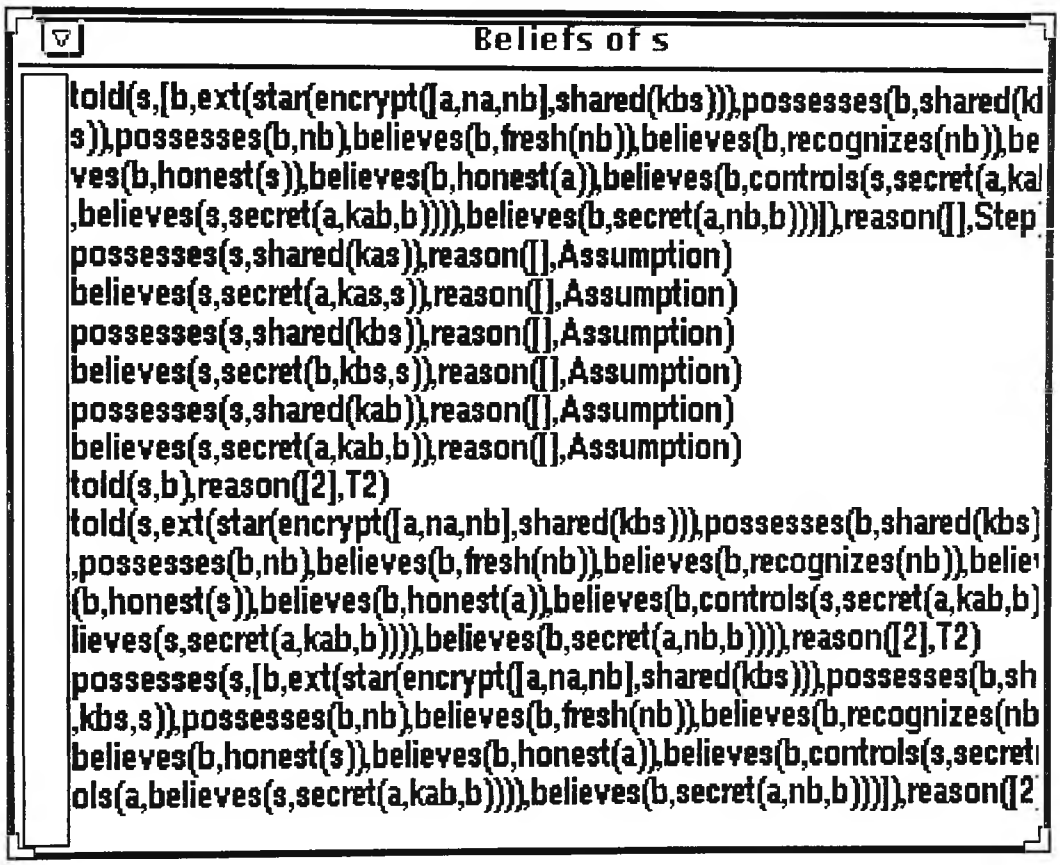

Beliefs of a
possesses(a,shared(kas)),reason([],Assumption)
believes(a,secret(a,kas,s)),reason([],Assumption)
possesses(a,na),reason([],Assumption)
believes(a,fresh(na)),reason([],Assumption)
believes(a,recognizes(b)),reason([],Assumption)
believes(a,honest(s)),reason([],Assumption)
believes(a,controls(s,secret(a,kab,b))),reason([],Assumpt
ion)

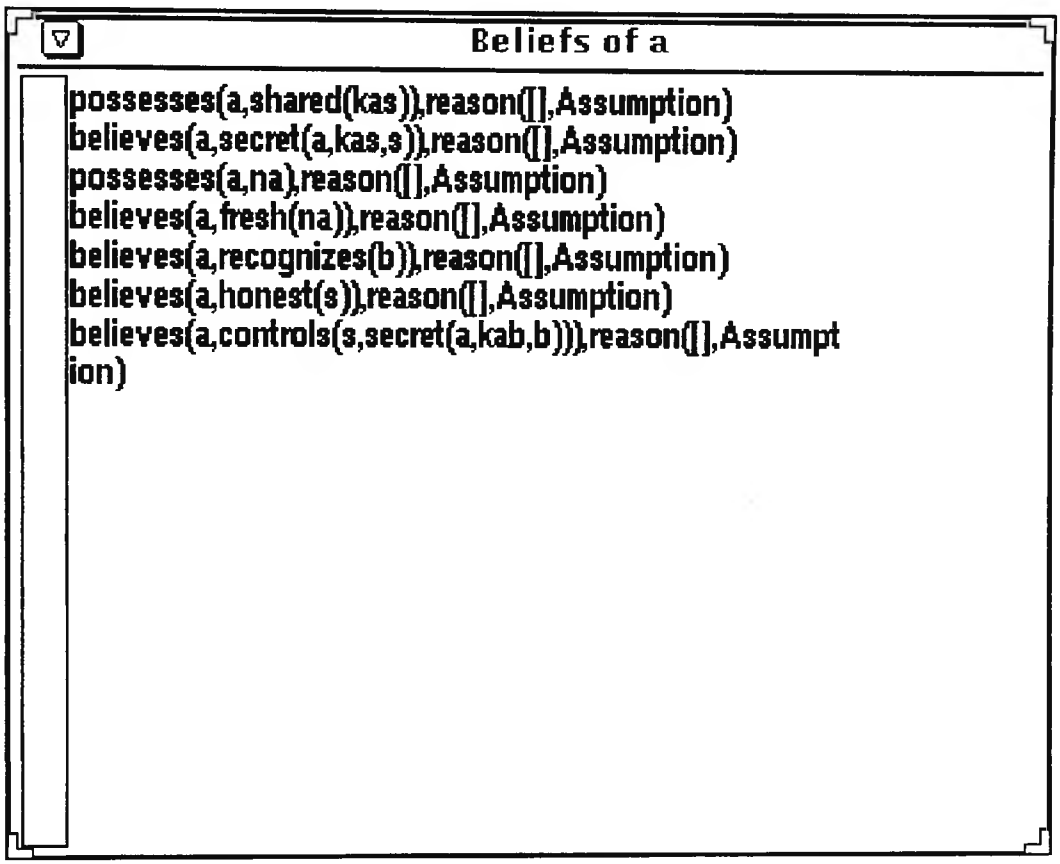

Beliefs of b
told(b,[a,na]),reason([],Step)
possesses(b,shared(kbs)),reason([],Assumption)
believes(b,secret(b,kbs,s)),reason([],Assumption)
possesses(b,nb),reason([],Assumption)
believes(b,fresh(nb)),reason([],Assumption)
believes(b,recognizes(nb)),reason([],Assumption)
believes(b,recognizes(a)),reason([],Assumption)
believes(b,honest(s)),reason([],Assumption)
believes(b,honest(a)),reason([],Assumption)
believes(b,controls(s,secret(a,kab,b))),reason([],Assumption)
believes(b,controls(a,believes(s,secret(a,kab,b)))),reason([],Assumption
believes(b,secret(a,nb,b)),reason([],Assumption)
told(b,a),reason([1],T2)
told(b,na),reason([1],T2)
possesses(b,[a,na]),reason([1],P1)
possesses(b,a),reason([29],P1)
possesses(b,na),reason([30],P1)
believes(b,recognizes([a,na])),reason([17,31],R11)

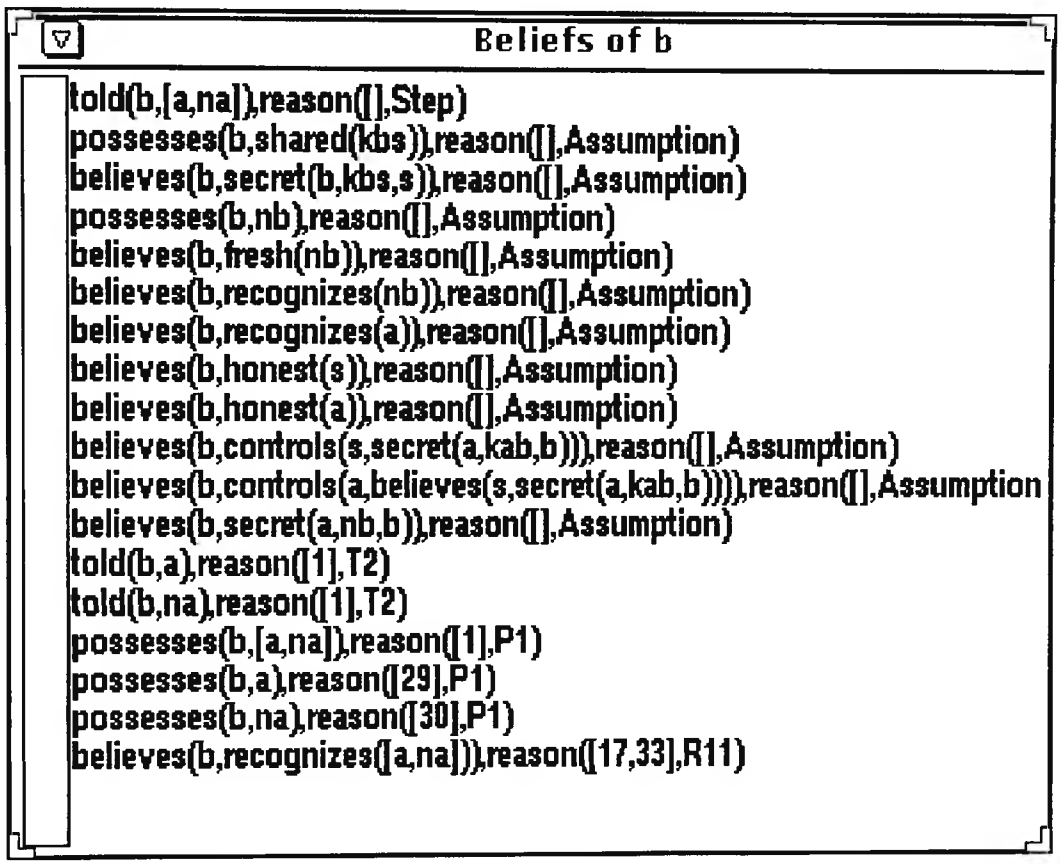

Beliefs of s
possesses(s, shared(kas)), reason([], 'Assumption')
believes(s, secret(a, kas, s)), reason([], 'Assumption')
possesses(s, shared(kbs)), reason([], 'Assumption')
believes(s, secret(b, kbs, s)), reason([], 'Assumption')
possesses(s, shared(kab)), reason([], 'Assumption')
believes(s, secret(a, kab, b)), reason([], 'Assumption')

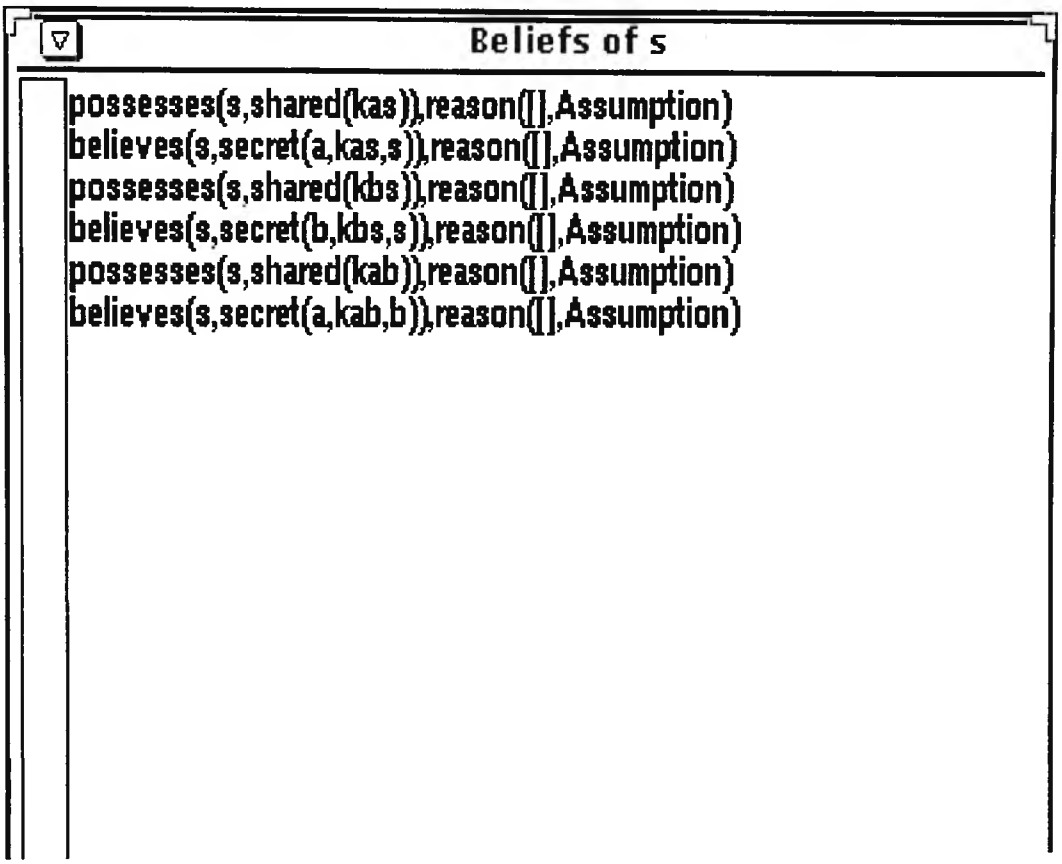

Beliefs of a
possesses(a, shared(kas)), reason([], 'Assumption')
believes(a, secret(a, kas, s)), reason([], 'Assumption')
possesses(a, na), reason([], 'Assumption')
believes(a, fresh(na)), reason([], 'Assumption')
believes(a, recognizes(b)), reason([], 'Assumption')
believes(a, honest(s)), reason([], 'Assumption')
believes(a, controls(s, secret(a, kab, b))), reason([], '
Assumption')


Beliefs of b
possesses(b, shared(kbs)), reason([], 'Assumption')
believes(b, secret(b, kbs, s)), reason([], 'Assumption')
possesses(b, nb), reason([], 'Assumption')
believes(b, fresh(nb)), reason([], 'Assumption')
believes(b, recognizes(nb)), reason([], 'Assumption')
believes(b, recognizes(a)), reason([], 'Assumption')
believes(b, honest(s)), reason([], 'Assumption')
believes(b, honest(a)), reason([], 'Assumption')
believes(b, controls(s, secret(a, kab, b))), reason([], '
Assumption')
believes(b, controls(a, believes(s, secret(a, kab, b)))),
reason([], 'Assumption')
believes(b, secret(a, nb, b)), reason([], 'Assumption')

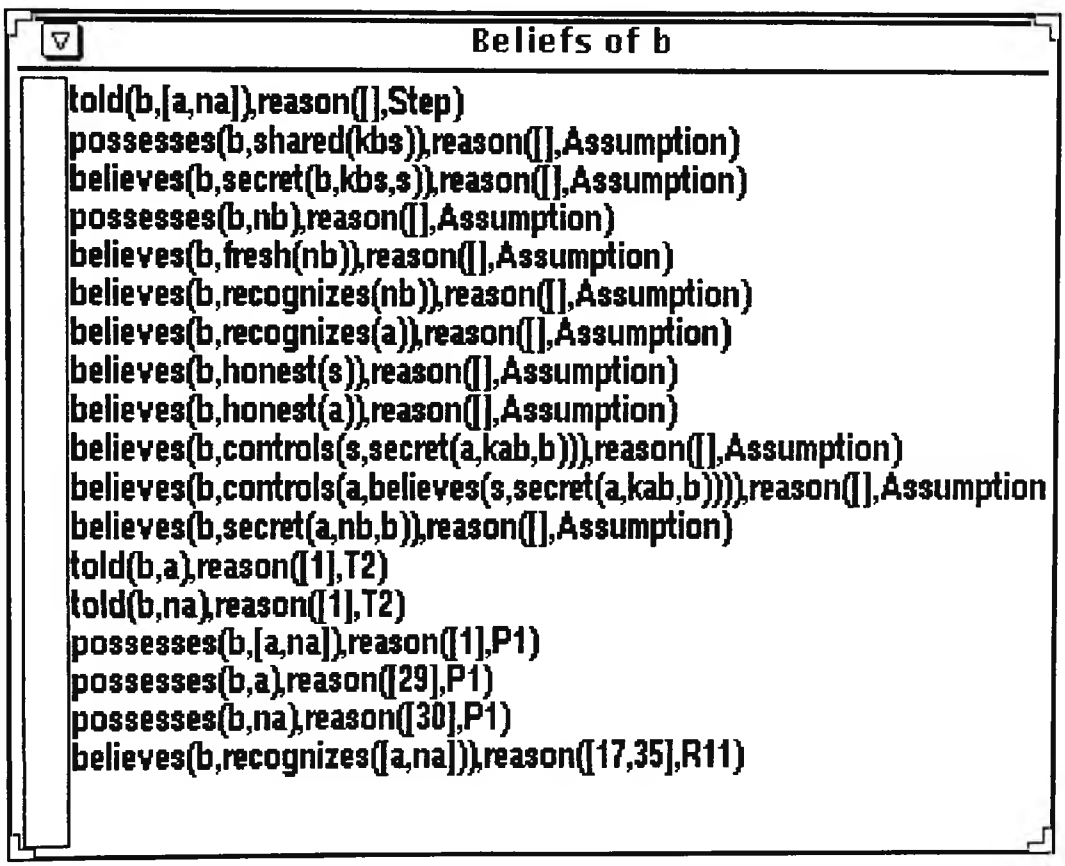

Beliefs of s
told(s,[b,ext(star(encrypt([a,na,nb],shared(kbs))),possesses(b,shared(kb
s)),possesses(b,nb),believes(b,fresh(nb)),believes(b,recognizes(nb)),be
ves(b,honest(s)),believes(b,honest(a)),believes(b,controls(s,secret(a,ka
,believes(s,secret(a,kab,b)))),believes(b,secret(a,nb,b)))]),reason([],Step
possesses(s,shared(kas)),reason([],Assumption)
believes(s,secret(a,kas,s)),reason([],Assumption)
possesses(s,shared(kbs)),reason([],Assumption)
believes(s,secret(b,kbs,s)),reason([],Assumption)
possesses(s,shared(kab)),reason([],Assumption)
believes(s,secret(a,kab,b)),reason([],Assumption)
told(s,b),reason([2],T2)
told(s,ext(star(encrypt([a,na,nb],shared(kbs))),possesses(b,shared(kbs)
,possesses(b,nb),believes(b,fresh(nb)),believes(b,recognizes(nb)),belie
(b,honest(s)),believes(b,honest(a)),believes(b,controls(s,secret(a,kab,b)
lieves(s,secret(a,kab,b)))),believes(b,secret(a,nb,b)))),reason([2],T2)
possesses(s,[b,ext(star(encrypt([a,na,nb],shared(kbs))),possesses(b,sh
,kbs,s)),possesses(b,nb),believes(b,fresh(nb)),believes(b,recognizes(nb
believes(b,honest(s)),believes(b,honest(a)),believes(b,controls(s,secret
ols(a,believes(s,secret(a,kab,b)))),believes(b,secret(a,nb,b)))]),reason([2

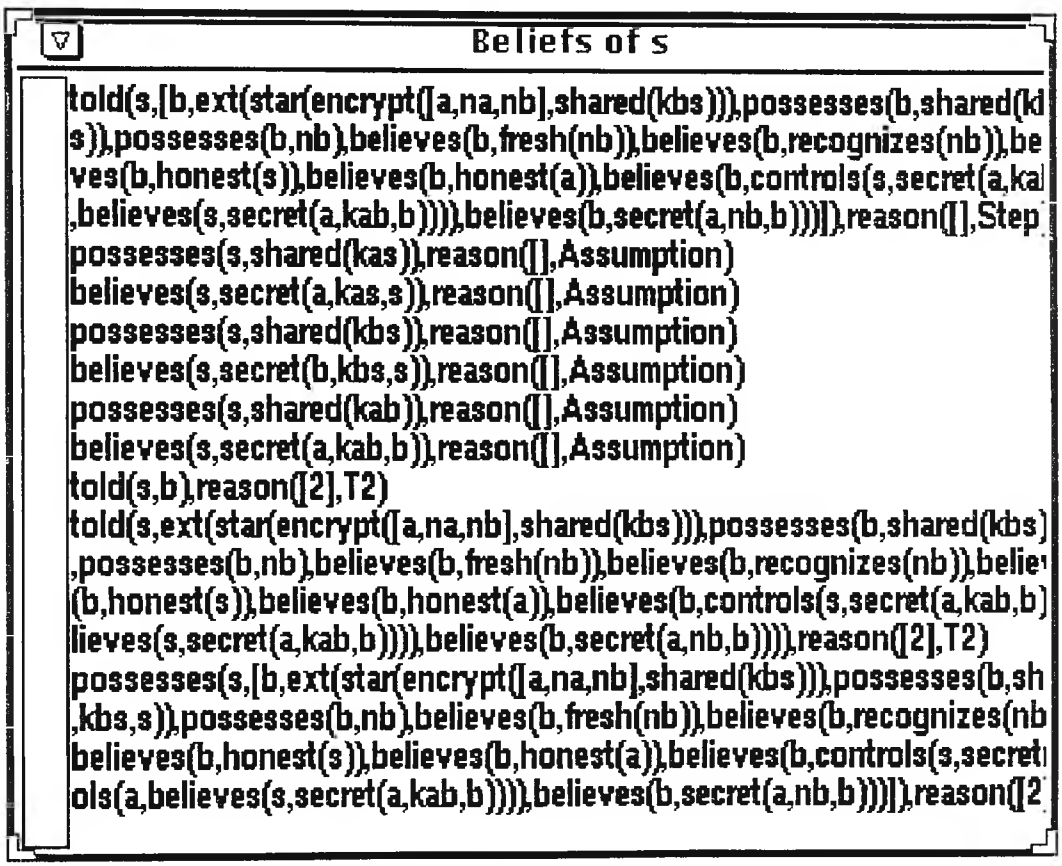

Beliefs of a
possesses(a,shared(kas)),reason([],Assumption)
believes(a,secret(a,kas,s)),reason([],Assumption)
possesses(a,na),reason([],Assumption)
believes(a,fresh(na)),reason([],Assumption)
believes(a,recognizes(b)),reason([],Assumption)
believes(a,honest(s)),reason([],Assumption)
believes(a,controls(s,secret(a,kab,b))),reason([],Assumpt
ion)

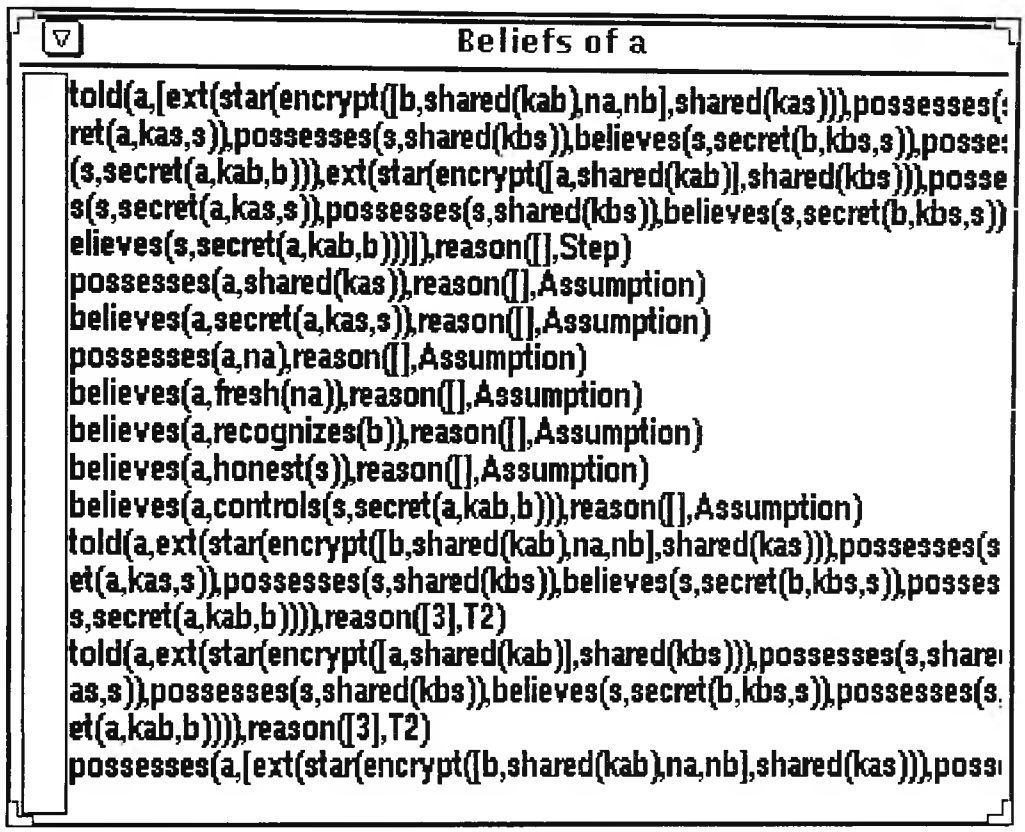

Beliefs of b
told(b,[a,na]),reason([],Step)
possesses(b,shared(kbs)),reason([],Assumption)
believes(b,secret(b,kbs,s)),reason([],Assumption)
possesses(b,nb),reason([],Assumption)
believes(b,fresh(nb)),reason([],Assumption)
believes(b,recognizes(nb)),reason([],Assumption)
believes(b,recognizes(a)),reason([],Assumption)
believes(b,honest(s)),reason([],Assumption)
believes(b,honest(a)),reason([],Assumption)
believes(b,controls(s,secret(a,kab,b))),reason([],Assumption)
believes(b,controls(a,believes(s,secret(a,kab,b)))),reason([],Assumption
believes(b,secret(a,nb,b)),reason([],Assumption)
told(b,a),reason([1],T2)
told(b,na),reason([1],T2)
possesses(b,[a,na]),reason([1],P1)
possesses(b,a),reason([29],P1)
possesses(b,na),reason([30],P1)
believes(b,recognizes([a,na])),reason([17,33],R11)

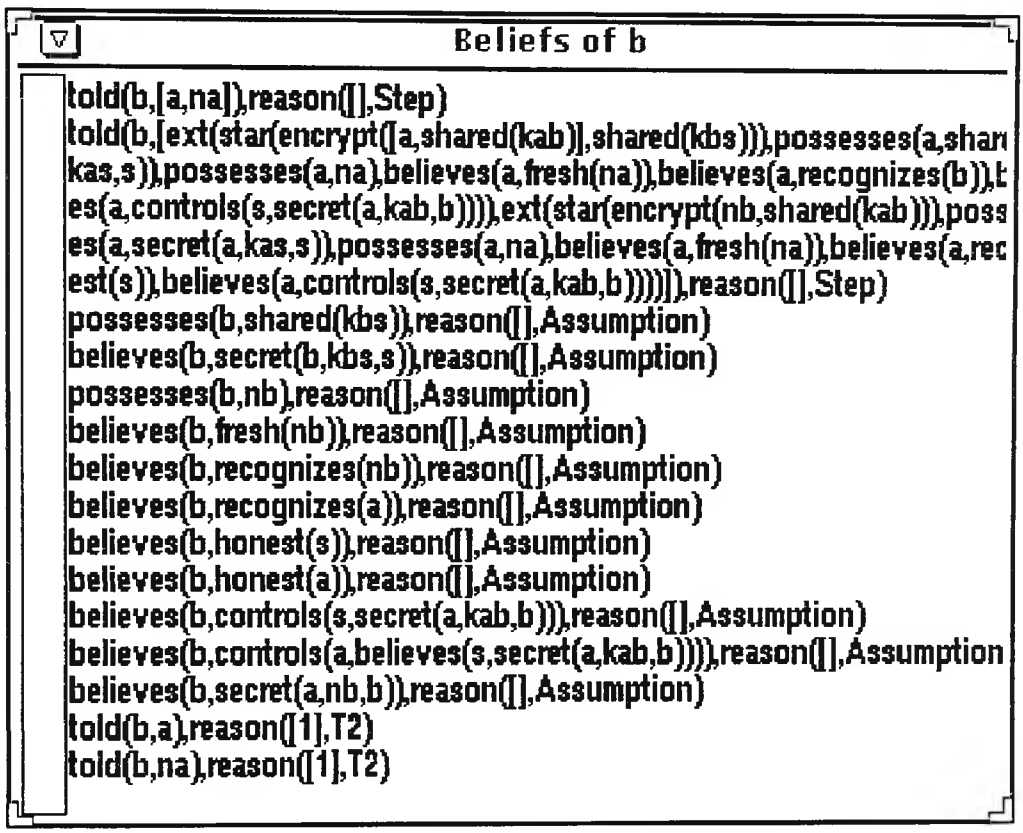

Beliefs of s
possesses(s,shared(kas)),reason([],Assumption)
believes(s,secret(a,kas,s)),reason([],Assumption)
possesses(s,shared(kbs)),reason([],Assumption)
believes(s,secret(b,kbs,s)),reason([],Assumption)
possesses(s,shared(kab)),reason([],Assumption)
believes(s,secret(a,kab,b)),reason([],Assumption)

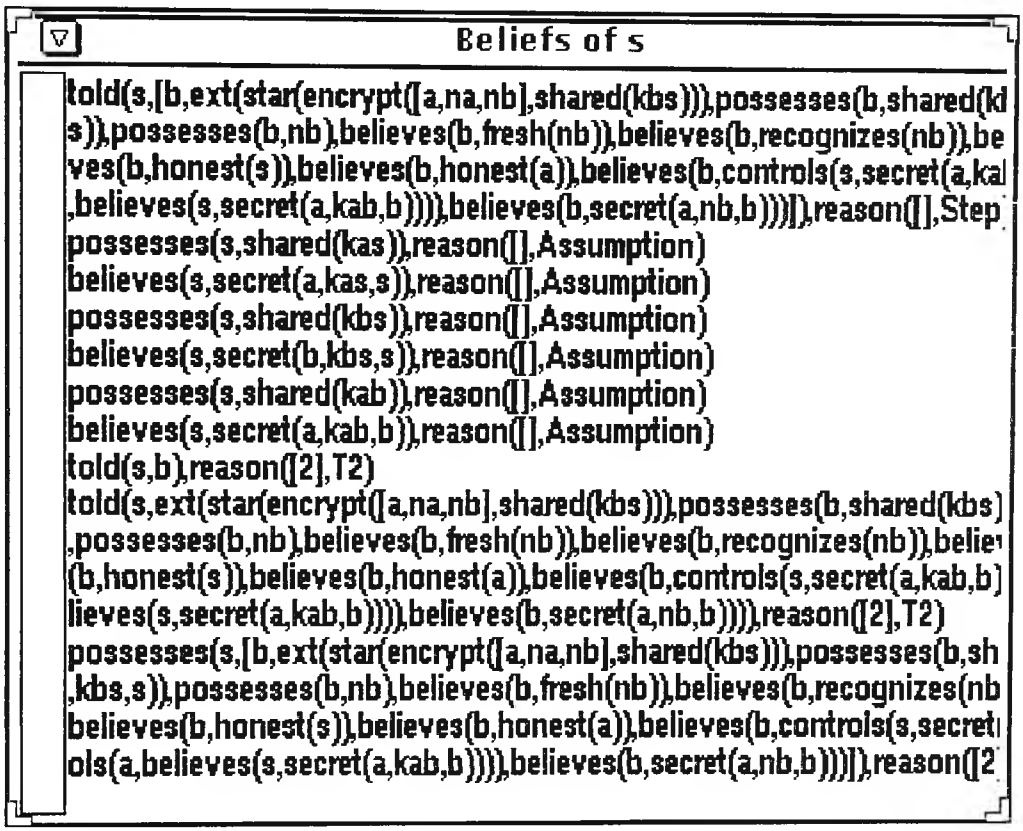

Beliefs of a
told(a,[ext(star(encrypt([b,shared(kab),na,nb],shared(kas))),possesses(
ret(a,kas,s)),possesses(s,shared(kbs)),believes(s,secret(b,kbs,s)),posse
(s,secret(a,kab,b))),ext(star(encrypt([a,shared(kab)],shared(kbs))),posse
s(s,secret(a,kas,s)),possesses(s,shared(kbs)),believes(s,secret(b,kbs,s))
elieves(s,secret(a,kab,b)))]),reason([],Step)
possesses(a,shared(kas)),reason([],Assumption)
believes(a,secret(a,kas,s)),reason([],Assumption)
possesses(a,na),reason([],Assumption)
believes(a,fresh(na)),reason([],Assumption)
believes(a,recognizes(b)),reason([],Assumption)
believes(a,honest(s)),reason([],Assumption)
believes(a,controls(s,secret(a,kab,b))),reason([],Assumption)
told(a,ext(star(encrypt([b,shared(kab),na,nb],shared(kas))),possesses(s
et(a,kas,s)),possesses(s,shared(kbs)),believes(s,secret(b,kbs,s)),posses
s,secret(a,kab,b)))),reason([3],T2)
told(a,ext(star(encrypt([a,shared(kab)],shared(kbs))),possesses(s,share
as,s)),possesses(s,shared(kbs)),believes(s,secret(b,kbs,s)),possesses(s,
et(a,kab,b)))),reason([3],T2)
possesses(a,[ext(star(encrypt([b,shared(kab),na,nb],shared(kas))),poss

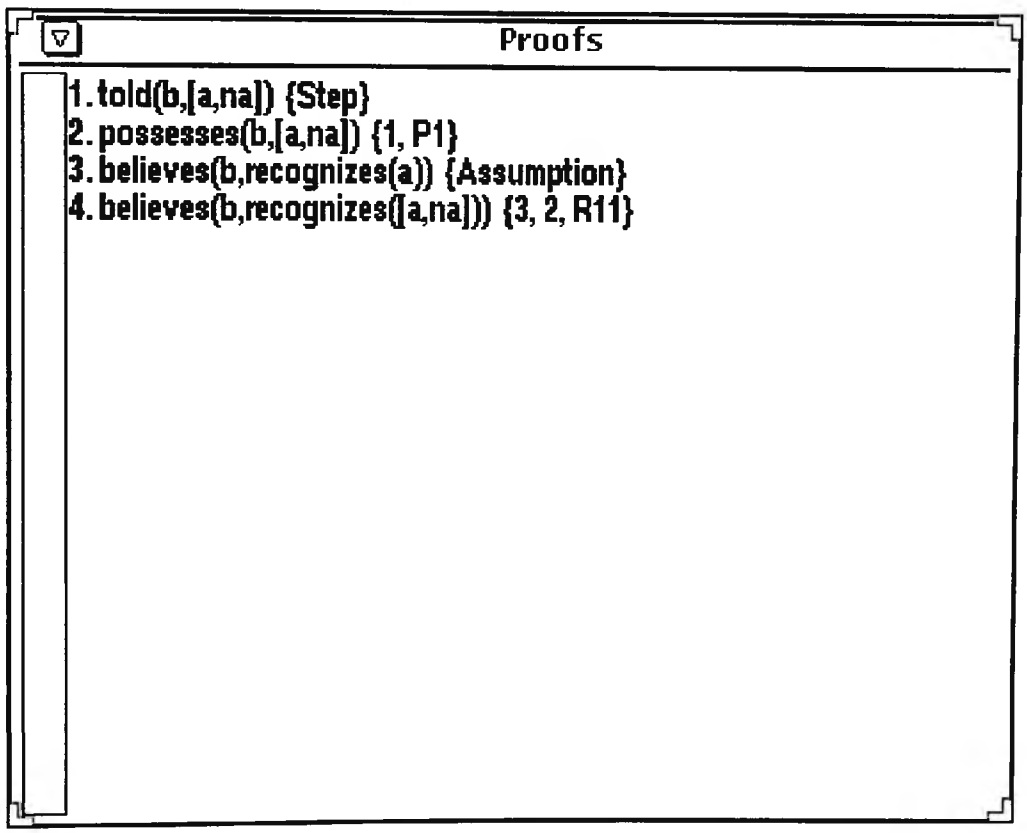

Beliefs of b
told(b,[a,na]),reason([],Step)
possesses(b,shared(kbs)),reason([],Assumption)
believes(b,secret(b,kbs,s)),reason([],Assumption)
possesses(b,nb),reason([],Assumption)
believes(b,fresh(nb)),reason([],Assumption)
believes(b,recognizes(nb)),reason([],Assumption)
believes(b,recognizes(a)),reason([],Assumption)
believes(b,honest(s)),reason([],Assumption)
believes(b,honest(a)),reason([],Assumption)
believes(b,controls(s,secret(a,kab,b))),reason([],Assumption)
believes(b,controls(a,believes(s,secret(a,kab,b)))),reason([],Assumption
believes(b,secret(a,nb,b)),reason([],Assumption)
told(b,a),reason([1],T2)
told(b,na),reason([1],T2)
possesses(b,[a,na]),reason([1],P1)
possesses(b,a),reason([29],P1)
possesses(b,na),reason([30],P1)
believes(b,recognizes([a,na])),reason([17,35],R11)


Beliefs of s
told(s,[b,ext(star(encrypt([a,na,nb],shared(kbs))),possesses(b,shared(kb
s)),possesses(b,nb),believes(b,fresh(nb)),believes(b,recognizes(nb)),be
ves(b,honest(s)),believes(b,honest(a)),believes(b,controls(s,secret(a,ka
,believes(s,secret(a,kab,b)))),believes(b,secret(a,nb,b)))]),reason([],Step
possesses(s,shared(kas)),reason([],Assumption)
believes(s,secret(a,kas,s)),reason([],Assumption)
possesses(s,shared(kbs)),reason([],Assumption)
believes(s,secret(b,kbs,s)),reason([],Assumption)
possesses(s,shared(kab)),reason([],Assumption)
believes(s,secret(a,kab,b)),reason([],Assumption)
told(s,b),reason([2],T2)
told(s,ext(star(encrypt([a,na,nb],shared(kbs))),possesses(b,shared(kbs)
,possesses(b,nb),believes(b,fresh(nb)),believes(b,recognizes(nb)),belie
(b,honest(s)),believes(b,honest(a)),believes(b,controls(s,secret(a,kab,b)
lieves(s,secret(a,kab,b)))),believes(b,secret(a,nb,b)))),reason([2],T2)
possesses(s,[b,ext(star(encrypt([a,na,nb],shared(kbs))),possesses(b,sh
,kbs,s)),possesses(b,nb),believes(b,fresh(nb)),believes(b,recognizes(nb
believes(b,honest(s)),believes(b,honest(a)),believes(b,controls(s,secret
ols(a,believes(s,secret(a,kab,b)))),believes(b,secret(a,nb,b)))]),reason([2


Beliefs of a
told(a,[ext(star(encrypt([b,shared(kab),na,nb],shared(kas))),possesses(s
ret(a,kas,s)),possesses(s,shared(kbs)),believes(s,secret(b,kbs,s)),posse
(s,secret(a,kab,b))),ext(star(encrypt([a,shared(kab)],shared(kbs))),posse
s(s,secret(a,kas,s)),possesses(s,shared(kbs)),believes(s,secret(b,kbs,s))
elieves(s,secret(a,kab,b)))]),reason([],Step)
possesses(a,shared(kas)),reason([],Assumption)
believes(a,secret(a,kas,s)),reason([],Assumption)
possesses(a,na),reason([],Assumption)
believes(a,fresh(na)),reason([],Assumption)
believes(a,recognizes(b)),reason([],Assumption)
believes(a,honest(s)),reason([],Assumption)
believes(a,controls(s,secret(a,kab,b))),reason([],Assumption)
told(a,ext(star(encrypt([b,shared(kab),na,nb],shared(kas))),possesses(s
et(a,kas,s)),possesses(s,shared(kbs)),believes(s,secret(b,kbs,s)),posses
s,secret(a,kab,b)))),reason([3],T2)
told(a,ext(star(encrypt([a,shared(kab)],shared(kbs))),possesses(s,share
as,s)),possesses(s,shared(kbs)),believes(s,secret(b,kbs,s)),possesses(s,
et(a,kab,b)))),reason([3],T2)
possesses(a,[ext(star(encrypt([b,shared(kab),na,nb],shared(kas))),poss


Beliefs of b
told(b,[a,na]),reason([],Step)
told(b,[ext(star(encrypt([a,shared(kab)],shared(kbs))),possesses(a,shar
kas,s)),possesses(a,na),believes(a,fresh(na)),believes(a,recognizes(b)),t
es(a,controls(s,secret(a,kab,b)))),ext(star(encrypt(nb,shared(kab))),poss
es(a,secret(a,kas,s)),possesses(a,na),believes(a,fresh(na)),believes(a,rec
est(s)),believes(a,controls(s,secret(a,kab,b))))]),reason([],Step)
possesses(b,shared(kbs)),reason([],Assumption)
believes(b,secret(b,kbs,s)),reason([],Assumption)
possesses(b,nb),reason([],Assumption)
believes(b,fresh(nb)),reason([],Assumption)
believes(b,recognizes(nb)),reason([],Assumption)
believes(b,recognizes(a)),reason([],Assumption)
believes(b,honest(s)),reason([],Assumption)
believes(b,honest(a)),reason([],Assumption)
believes(b,controls(s,secret(a,kab,b))),reason([],Assumption)
believes(b,controls(a,believes(s,secret(a,kab,b)))),reason([],Assumption
believes(b,secret(a,nb,b)),reason([],Assumption)
told(b,a),reason([1],T2)
told(b,na),reason([1],T2)


Beliefs of s
told(s,[b,ext(star(encrypt([a,na,nb],shared(kbs))),possesses(b,shared(kb
s)),possesses(b,nb),believes(b,fresh(nb)),believes(b,recognizes(nb)),be
ves(b,honest(s)),believes(b,honest(a)),believes(b,controls(s,secret(a,kal
,believes(s,secret(a,kab,b)))),believes(b,secret(a,nb,b)))]),reason([],Step
possesses(s,shared(kas)),reason([],Assumption)
believes(s,secret(a,kas,s)),reason([],Assumption)
possesses(s,shared(kbs)),reason([],Assumption)
believes(s,secret(b,kbs,s)),reason([],Assumption)
possesses(s,shared(kab)),reason([],Assumption)
believes(s,secret(a,kab,b)),reason([],Assumption)
told(s,b),reason([2],T2)
told(s,ext(star(encrypt([a,na,nb],shared(kbs))),possesses(b,shared(kbs)
,possesses(b,nb),believes(b,fresh(nb)),believes(b,recognizes(nb)),belie
(b,honest(s)),believes(b,honest(a)),believes(b,controls(s,secret(a,kab,b)
lieves(s,secret(a,kab,b)))),believes(b,secret(a,nb,b)))),reason([2],T2)
possesses(s,[b,ext(star(encrypt([a,na,nb],shared(kbs))),possesses(b,sh
,kbs,s)),possesses(b,nb),believes(b,fresh(nb)),believes(b,recognizes(nb
believes(b,honest(s)),believes(b,honest(a)),believes(b,controls(s,secret
ols(a,believes(s,secret(a,kab,b)))),believes(b,secret(a,nb,b)))]),reason([2


Proofs
1. told(b,[a,na]) {Step}
2. possesses(b,[a,na]) {1, P1}
3. believes(b,recognizes(a)) {Assumption}
4. believes(b,recognizes([a,na])) {3, 2, R11}

B.3 Output Screens

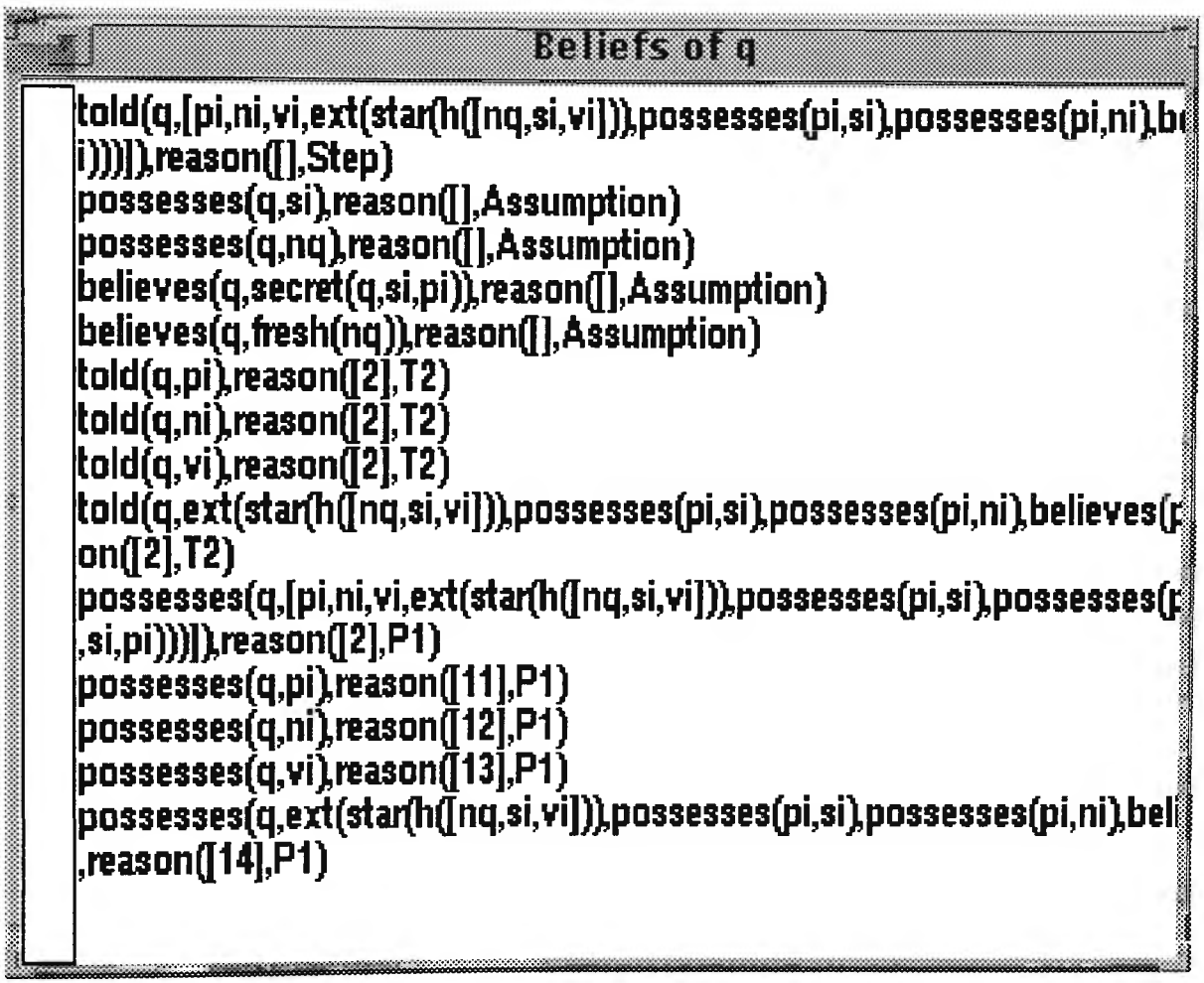
The following pages show the output screens for the beliefs subwindows obtained from the *Protocol Analyzer* for experiments with the Voting and Otway-Rees protocols.

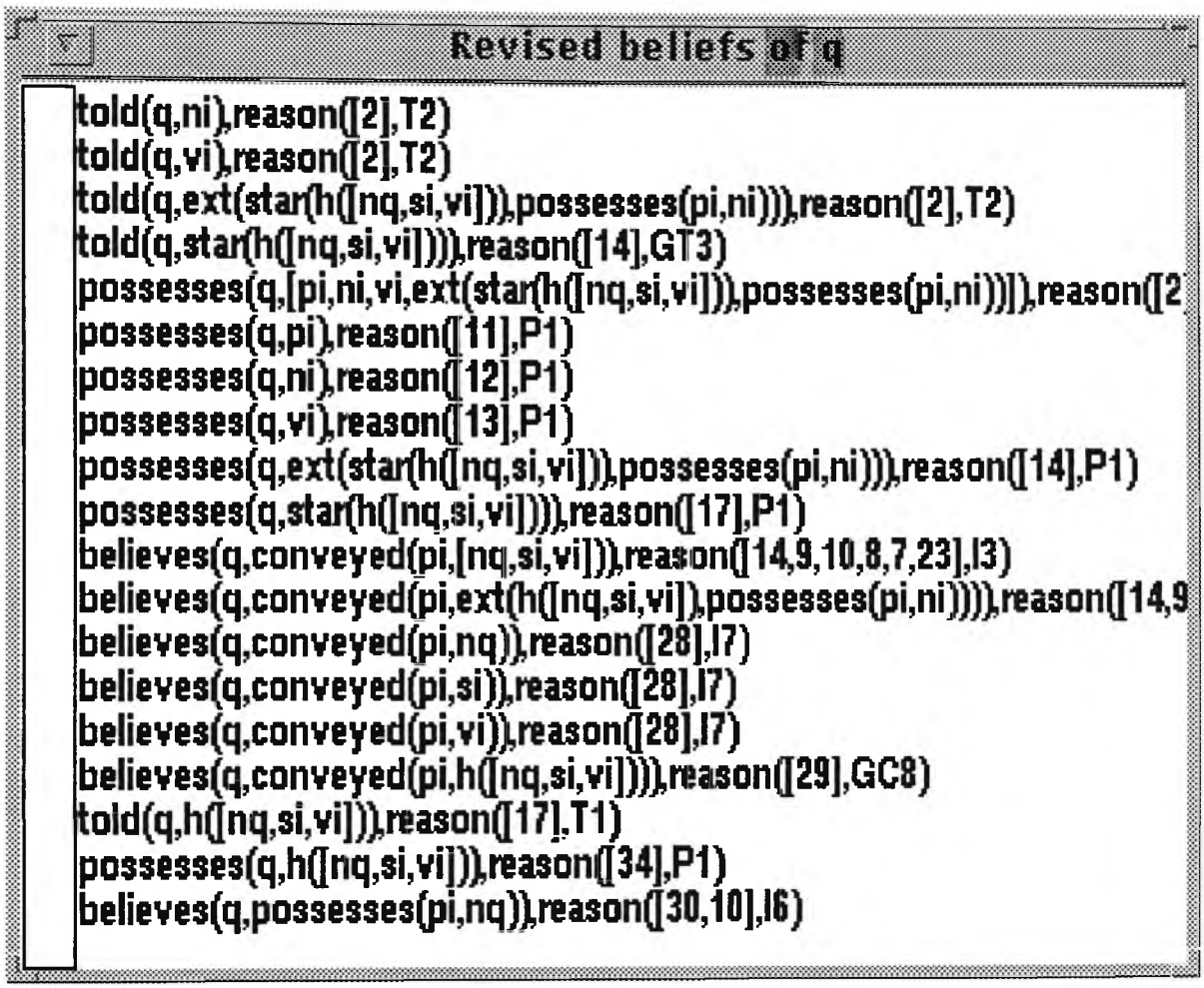
Protocol Analyzer with the Voting protocol:

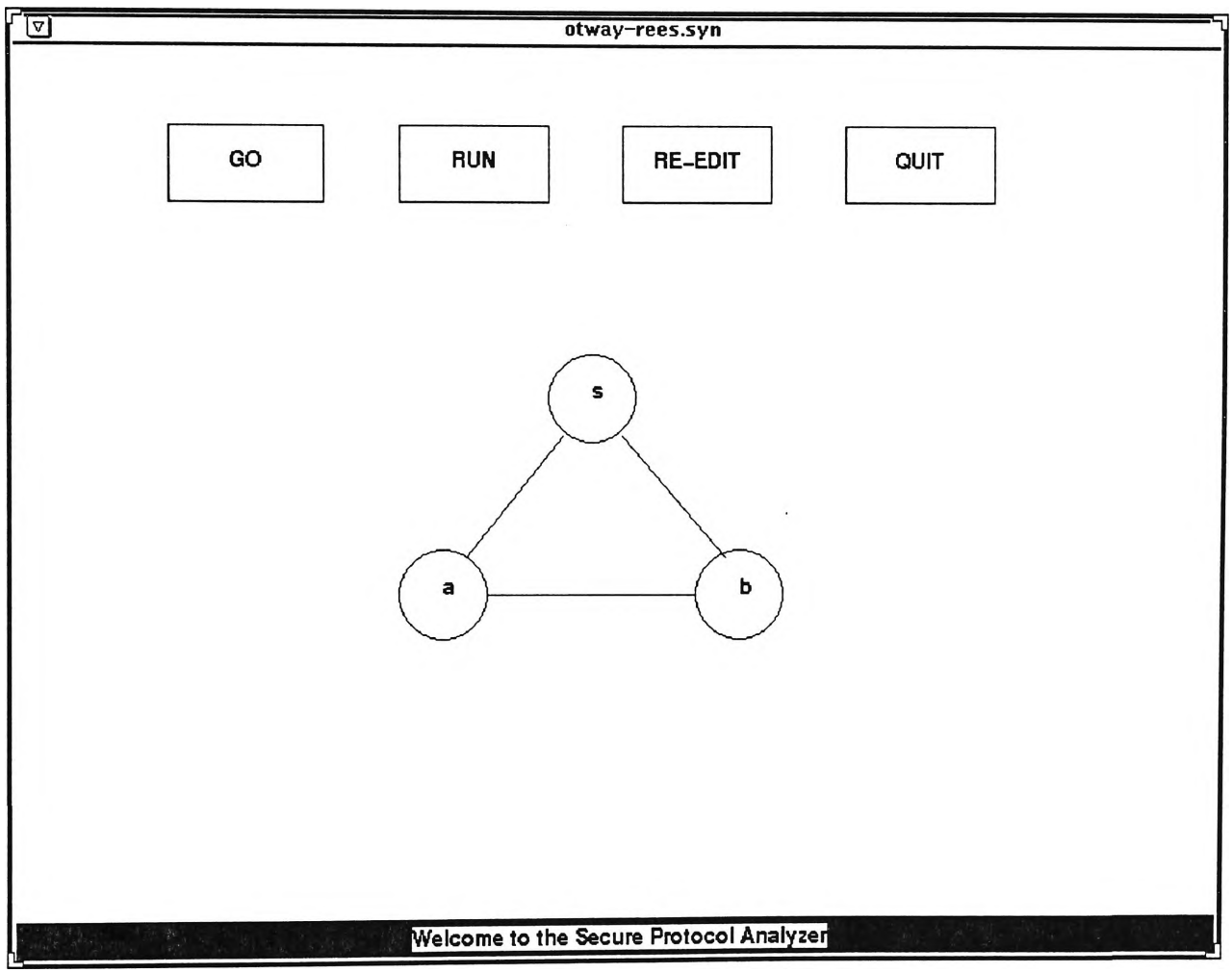
1. Beliefs subwindow showing q's beliefs after the run of the protocol with the original set of initial assumptions
2. Beliefs subwindow showing q's beliefs after the run of the protocol with the initial assumptions, modified as given in Chapter 6, subsection 6.2.1

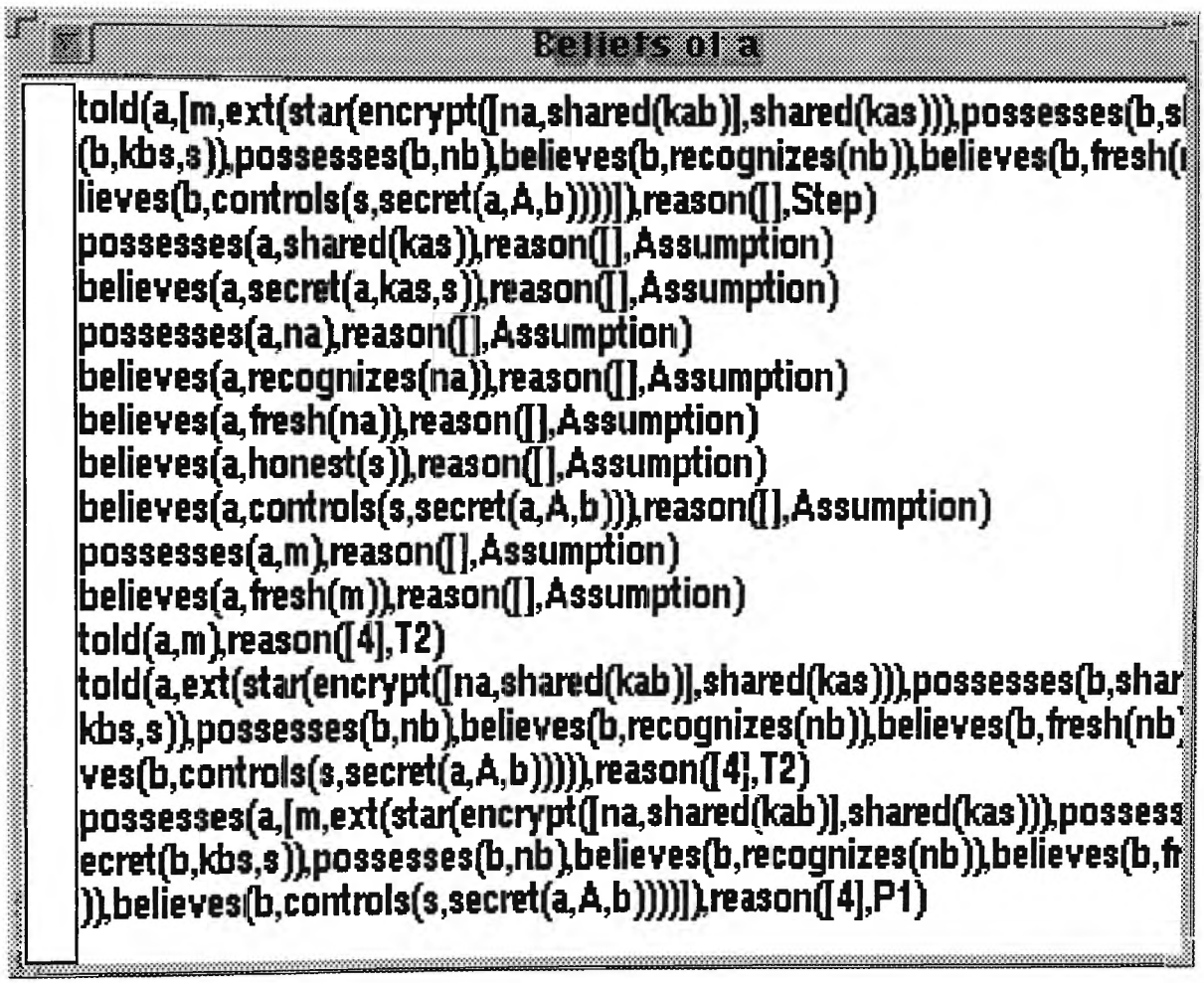
Protocol Analyzer with the Otway-Rees protocol:

1. Main window showing the protocol schematic diagram of Otway-Rees protocol
2. Beliefs subwindow showing a's beliefs after the run of the protocol with the original set of initial assumptions
3. Beliefs subwindow showing a's beliefs after the run of the protocol with initial assumptions, modified as given in Chapter 6, section 6.3


Beliefs of q
told(q,[pi,ni,vi,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),be
i))))]),reason([],Step)
possesses(q,si),reason([],Assumption)
possesses(q,nq),reason([],Assumption)
believes(q,secret(q,si,pi)),reason([],Assumption)
believes(q,fresh(nq)),reason([],Assumption)
told(q,pi),reason([2],T2)
told(q,ni),reason([2],T2)
told(q,vi),reason([2],T2)
told(q,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),believes(p
on([2],T2)
possesses(q,[pi,ni,vi,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(p
,si,pi))))]),reason([2],P1)
possesses(q,pi),reason([11],P1)
possesses(q,ni),reason([12],P1)
possesses(q,vi),reason([13],P1)
possesses(q,ext(star(h([nq,si,vi])),possesses(pi,si),possesses(pi,ni),bel
,reason([14],P1)


Revised beliefs of q
told(q,ni),reason([2],T2)
told(q,vi),reason([2],T2)
told(q,ext(star(h([nq,si,vi])),possesses(pi,ni))),reason([2],T2)
told(q,star(h([nq,si,vi]))),reason([14],GT3)
possesses(q,[pi,ni,vi,ext(star(h([nq,si,vi])),possesses(pi,ni))]),reason([2
possesses(q,pi),reason([11],P1)
possesses(q,ni),reason([12],P1)
possesses(q,vi),reason([13],P1)
possesses(q,ext(star(h([nq,si,vi])),possesses(pi,ni))),reason([14],P1)
possesses(q,star(h([nq,si,vi]))),reason([17],P1)
believes(q,conveyed(pi,[nq,si,vi])),reason([14,9,10,8,7,23],I3)
believes(q,conveyed(pi,ext(h([nq,si,vi]),possesses(pi,ni)))),reason([14,9
believes(q,conveyed(pi,nq)),reason([28],I7)
believes(q,conveyed(pi,si)),reason([28],I7)
believes(q,conveyed(pi,vi)),reason([28],I7)
believes(q,conveyed(pi,h([nq,si,vi]))),reason([29],GC8)
told(q,h([nq,si,vi])),reason([17],T1)
possesses(q,h([nq,si,vi])),reason([34],P1)
believes(q,possesses(pi,nq)),reason([30,10],I6)


otway-rees.syn
GO
RUN
RE-EDIT
QUIT
s
a
b
Welcome to the Secure Protocol Analyzer


Beliefs of a
told(a,[m,ext(star(encrypt([na,shared(kab)],shared(kas))),possesses(b,s
(b,kbs,s)),possesses(b,nb),believes(b,recognizes(nb)),believes(b,fresh(
lieves(b,controls(s,secret(a,A,b)))))]),reason([],Step)
possesses(a,shared(kas)),reason([],Assumption)
believes(a,secret(a,kas,s)),reason([],Assumption)
possesses(a,na),reason([],Assumption)
believes(a,recognizes(na)),reason([],Assumption)
believes(a,fresh(na)),reason([],Assumption)
believes(a,honest(s)),reason([],Assumption)
believes(a,controls(s,secret(a,A,b))),reason([],Assumption)
possesses(a,m),reason([],Assumption)
believes(a,fresh(m)),reason([],Assumption)
told(a,m),reason([4],T2)
told(a,ext(star(encrypt([na,shared(kab)],shared(kas))),possesses(b,shar
kbs,s)),possesses(b,nb),believes(b,recognizes(nb)),believes(b,fresh(nb
ves(b,controls(s,secret(a,A,b))))),reason([4],T2)
possesses(a,[m,ext(star(encrypt([na,shared(kab)],shared(kas))),possess
ecret(b,kbs,s)),possesses(b,nb),believes(b,recognizes(nb)),believes(b,f
)),believes(b,controls(s,secret(a,A,b)))))]),reason([4],P1)


Revised beliefs of a
told(a,[m,ext(star(encrypt([na,shared(kab)],shared(kas))),possesses(b,s
(b,kbs,s)),possesses(b,nb),believes(b,recognizes(nb)),believes(b,fresh(
lieves(b,controls(s,secret(a,A,b)))))]),reason([],Step)
possesses(a,shared(kas)),reason([],Assumption)
believes(a,secret(a,kas,s)),reason([],Assumption)
possesses(a,na),reason([],Assumption)
believes(a,recognizes(na)),reason([],Assumption)
believes(a,fresh(na)),reason([],Assumption)
believes(a,honest(s)),reason([],Assumption)
believes(a,controls(s,secret(a,A,b))),reason([],Assumption)
possesses(a,m),reason([],Assumption)
told(a,m),reason([4],T2)
told(a,ext(star(encrypt([na,shared(kab)],shared(kas))),possesses(b,shar
kbs,s)),possesses(b,nb),believes(b,recognizes(nb)),believes(b,fresh(nb
ves(b,controls(s,secret(a,A,b))))),reason([4],T2)
possesses(a,[m,ext(star(encrypt([na,shared(kab)],shared(kas))),possess
ecret(b,kbs,s)),possesses(b,nb),believes(b,recognizes(nb)),believes(b,fr
)),believes(b,controls(s,secret(a,A,b)))))]),reason([4],P1)
possesses(a,ext(star(encrypt([na,shared(kab)],shared(kas))),possesses(